Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Implementierung und Evaluation verschiedener Bayes-Filter für das Personentracking

Bachelorarbeit zur Erlangung des akademischen Grades Bachelor of Science

Robert Arenknecht

Betreuer: Michael Volkhardt
Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Die Bachelorarbeit wurde am 20.1.2015 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

urn:nbn:de:gbv:ilm1-2015200065

Erklärung: „Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet.“

Ilmenau, 20.1.2015 .....

Robert Arenknecht

Inhaltsverzeichnis

Kapitel 1

Einleitung

Für heutige mobile Roboterapplikationen ist es eine Kernaufgabe ihre Umwelt zu erkennen und auf sie zu reagieren. Dazu zählt neben der Lokalisierung und Hinderniserkennung beim Fahren auch die Erkennung von Personen und Objekten. Besonders die Erkennung von Personen steht bei den mobilen Robotern im Vordergrund, da Menschen der zentrale Kommunikations- und Interaktionspartner sind.
Die Aufgaben der Robotersysteme des Fachgebiets für Neuroinformatik und Kognitive Robotik (NIKR) haben dabei ganz verschiedene Aufgaben. Diese reichen von der Informationsauskunft und Navigation in Gebäuden wie in [Stricker et al., 2012], über den in [Gross et al., 2011] vorgestellten Roboter, welcher älteren Menschen zu Hause helfen und zu Bewegungstraining motivieren soll, bis hin zu einem mobilen Robotersystem aus [Gross et al., 2014], welches Schlaganfallpatienten bei ihrem Lauf- und Orientierungstraining in einer Klinik helfen soll. Obwohl diese drei Systeme sehr verschiedene Aufgaben haben, ist doch bei allen der Menschen der Interaktionspartner und müssen diesen in ihrer Umwelt wahrnehmen.

Das Gebiet der Personendetektion ist dabei ein sehr gut erforschtes Thema mit vielen verschiedenen Ansätzen und Lösungen. So werden Personen schon sehr gut durch Kameras, vgl. [Felzenszwalb et al., 2010], [Dalal und Triggs, 2005], oder auch Laserdistanzmessungen, vgl. [Arras et al., 2007], erkannt. Diese Personenerkennung sollte jedoch auch bei sich bewegenden Personen robust funktionieren.

An dieser Stelle stößt diese jedoch an ihre Grenzen. Durch Bewegungen der erkannten Personen kommt es häufig zu Verdeckungen, wie z.B. durch Gegenstände im Raum. Außerdem muss oft mit wechselnden Lichtverhältnissen gearbeitet werden. Um diese Engstellen der Personenerkennung zu umgehen, wird ein Personentrackingsystem verwendet. Dies kann verschiedene Detektoren miteinander verbinden und bei einem kurzen Ausfall dieser eine Personen-Hypothese aufrecht erhalten. Das heißt, dass Bewegungen von Personen simuliert werden, um so einen konstanten Kontakt zum Kommunikationspartner zu haben. Diese Simulation beruht auf dem Prinzip der Bayes-Filter in Zusammenhang mit einem Modell, welches die Bewegung einer Person beschreibt. Dieses Prinzip wird durch Kalman-, oder Partikel Filter realisiert. Diese schaffen durch ein Bewegungsmodell und den Messdaten der Personendetektion eine Brücke zwischen Simulation und Detektion.

1.1 Problemstellung

Die Bewegungen von Menschen zu simulieren ist eine schwierige Aufgabe, denn wir bewegen uns in unterschiedlichen Situationen auf verschiedene Arten. Diese verschiedenen Situationen müssen vom Tracker beachtet werden, damit die Bewegung stehts gut simuliert werden kann. Damit dies realisierbar ist, muss es möglich sein zur Laufzeit des Roboters die Bewegungsmodelle zu verändern, um auf verschiedene Situationen reagieren zu können.

Zur Zeit sind jedoch bei den verwendeten Methoden am FG NIKR Tracker und Modell fest miteinander verknüpft, sodass ein Wechsel nicht möglich ist. Außerdem gibt es verschiedene Kalman Filter, die mit unterschiedlichen Modellen unterschiedlich gut funktionieren. So ist es beim Standard Kalman Filter nicht möglich nichtlineare Bewegungen zu simulieren. Auch hier soll es möglich sein zwischen verschiedenen Filtern zu wechseln.

1.2 Ziele

Ziel dieser Arbeit soll es sein ein System zu implementieren, welches sowohl den Wechsel von Modellen, als auch den Wechsel von Filtern ermöglicht. Dafür wird eine Bibliothek an das bestehende MIRA Framework, siehe [Einhorn et al., 2012] angebunden, die genau diese Funktionen bietet. Bei der Anbindung ist es wichtig, dass sowohl die Standards der Bibliothek als auch der MIRA Umgebung eingehalten werden.

Neben der Bibliothek soll zu dem schon bestehenden auch ein weiteres Bewegungsmodell implementiert werden. Im Gegensatz zu dem bestehenden Modell, ist das Neue ein nichtlineares Bewegungsmodell, welches über die Position der Person im Zusammenspiel mit der Geschwindigkeit und Orientierung die Bewegung eines Menschen beschreibt.

1.3 Struktur

Zunächst werden in Kapitel 2 die theoretischen Grundlagen eines Kalman Filters und der Bewegungsmodelle erläutert. Dabei wird auf verschiedenen Variationen des Kalman Filter eingegangen und ein Überblick über das Personentracking gegeben.
Anschließend werden der aktuelle Stand in Kapitel 3 besprochen und sowohl die Bibliothek, welche angebunden wird, als auch das nichtlineare Modell, welches implementiert werden soll, vorgestellt.
Der genaue Ablauf und die Struktur dieser Implementierung sind in Kapitel 4 dargestellt. Es wird zunächst allgemein die Bibliothek an das bestehende MIRA Framework angebunden und anschließend die beiden Bewegungsmodelle implementiert.
Dieses System wird dann in Kapitel 5 getestet und die beiden Modelle miteinander verglichen, bevor in Kapitel 6 nochmals eine Zusammenfassung dieser Arbeit gegeben wird.

Kapitel 2

Theoretische Grundlagen

2.1 Wahrscheinlichkeit

Die Wahrscheinlichkeit beschreibt, wie sicher eine Ereignis in einem Zufallsexperiment eintritt. Dieses Maß reicht von 0 bis 1, wobei 0 absolut unsicher und 1 absolut sicher beschreibt.

Die Aussage A beschreibt dabei das Auftreten einer Zufallsvariablen in einem Zufallsexperiment, vgl. [Goehler, 2007, Seite 95], zum Beispiel das Würfeln der Zahl 6.

$$P(A) = \frac{Anzahl\ der\ für\ A\ günstigen\ Ereignisse}{Anzahl\ aller\ möglichen\ Ereignisse};\ P(6\ wird\ gewürfelt) = \frac{1}{6} \tag{2.1}$$

Man kann auch Wahrscheinlichkeiten von Aussagen unter bestimmten Bedingungen beschreiben also z.B. die Wahrscheinlichkeit von Aussage A mit der Vorgabe von Aussage B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2.2}$$

Wahrscheinlichkeiten können auch als Funktion dargestellt werden. Man spricht dann von einer Verteilungsdichte-Funktion. Die Wahrscheinlichkeit einer Aussage errechnet

sich über die Integration dieser Funktion entlang der Aussage, [GOEHLER, 2007, Seite 96].

$$P(A) = \int_a^b p(x)dx \tag{2.3}$$

Zu einer Wahrscheinlichkeitsverteilung kann man einen Erwartungswert E, im Folgenden auch μ, und seine Varianz σ ausrechnen. Der Erwartungswert E beschreibt die Zahl, die die Zufallsvariable im Mittel annimmt. Die Varianz σ beschreibt die Unsicherheit des Erwartungswertes, [GOEHLER, 2007, Seite 96].

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx \tag{2.4}$$

$$\sigma(x) = \int_{-\infty}^{\infty} (x - E(X))^2 f(x)dx \tag{2.5}$$

In einem mehrdimensionalen System ist der Erwartungswert ein Vektor $\boldsymbol{\mu}$ und die Varianz eine Kovarianz, im folgenden als $\boldsymbol{\Sigma}$ bezeichnet.

2.2 Gauß-Verteilung

Die Gauß-Verteilung, oder auch Normalverteilung, ist eine besondere Art der Wahrscheinlichkeitsverteilung, welche auf dem zentralen Grenzwertsatz beruht. Dieser besagt, dass die Summe einer großen Zahl von unabhängigen Zufallsvariablen asymptotisch einer stabilen Verteilung folgt. Ist die Varianz endlich und positiv, dann ist diese Verteilung eine Normalverteilung. Definiert ist die Normalverteilung durch die Verteilungsdichte, [GOEHLER, 2007, Seite 98]:

$$p(\mathbf{x}) = det(2\pi\boldsymbol{\Sigma})^{-\frac{1}{2}} exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-\mathbf{1}}(\mathbf{x} - \boldsymbol{\mu})\right\} \tag{2.6}$$

Die Normalverteilung lässt sich durch 2 Parameter beschreiben.

$\boldsymbol{\mu}$	–	Erwartungswert
σ bzw. $\boldsymbol{\Sigma}$	–	Varianz bzw. Kovarianz

Die Fläche unter einer Wahrscheinlichkeitsverteilung ist immer eins, weshalb die Varianz das 'Aussehen' der Funktion beschreibt, siehe Abbildung 2.1. Ist die Varianz klein, so ist die Gauß-Kurve schmal und hoch. Bei einer großen Varianz ist die Kurve dagegen flach und breit.

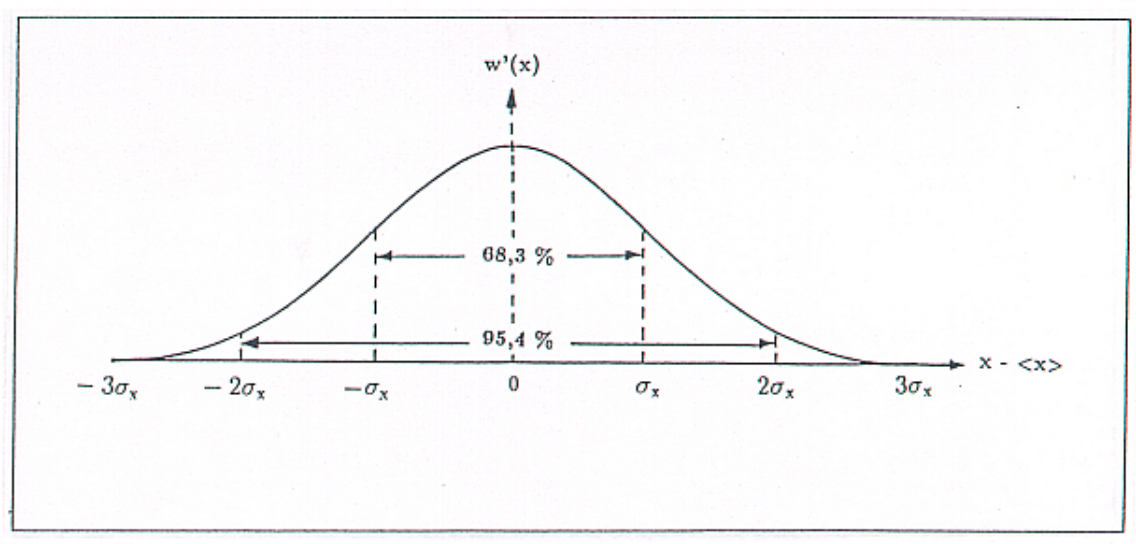

Abbildung 2.1: 1-D Normalverteilung, Quelle: [Weiprecht, 2012]
Graph eine Normalverteilung entlang der x-Achse. Der Erwartungswert μ liegt bei $x = 0$. Die breite des Graphen wird durch σ bestimmt. $68,3$ % der Fläche unter dem Graphen liegen zwischen $-\sigma$ und $+\sigma$. Das heißt, dass zu $68,3$ % Werte in diesem Bereich auftreten. Verdoppelt man diesen Bereich zu $2\,\sigma$, liegt die Wahrscheinlichkeit bei $95,4\%$.

2.3 Bayes-Filter Algorithmus

Der Bayes-Filter bietet eine Möglichkeit die Wahrscheinlichkeitsverteilung eines Systemzustands auszurechnen, [THRUN et al., 2006, Seite 26 ff.]. Der Systemzustand kann dabei nicht beobachtbare Zustandsgrößen haben, welche durch den Bayes-Filter geschätzt werden. Dazu werden Eingangs- und Messdaten verwendet und über mathematische Modelle mit dem Systemzustand verknüpft. Sie stellen also einen Zusammenhang zwischen dem Eingang des Systems u_t, bzw. den Messdaten z_t mit seinem Zustand x_t her.
Der Bayes-Filter arbeitet rekursiv über die Zeit. Die aktuelle Zustandsschätzung $bel(x_t)$ ergibt sich also aus der alten Schätzung $bel(x_{t-1})$ zusammen mit dem Eingang u_t und Ausgang z_t des Systems, siehe Abbildung 2.2.

Eingaben

1 $bel(x_{t-1})$ *// Zustandsschätzung des letzten Zeitschritts*

Algorithmus

2 `for all` x_t `do;` *// für alle möglichen neuen Zustände*
3 $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1}$; *// Berechnung der Wahrscheinlichkeit des Zustands x_t*
4 $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$; *// Korrektur der Wahrscheinlichkeit von x_t durch die Messung z_t*
5 `endfor;`

Rückgabe

6 $bel(x_t)$ *// Neue Schätzung*

Abbildung 2.2: Bayes-Filter Algorithmus

Die Berechnung für alle möglichen Zustände x_t erfolgt in zwei Schritten. In Zeile 3 wird zunächst die Wahrscheinlichkeit des neuen Zustands $\overline{bel}(x_t)$ über den Eingang u_t und der letzten Zustandswahrscheinlichkeit $bel(x_{t-1})$ ausgerechnet. In Zeile 4 wird dann

mit diesem Wert und mit der Messung z_t die Wahrscheinlichkeit $bel(x_t)$ ausgerechnet. Die mathematischen Modelle sind hierbei als Verteilungsdichte-Funktion dargestellt. Sie beschreiben die Wahrscheinlichkeit eines neuen Zustands unter der Bedingung des Eingangs, bzw. einer Messung und des alten geschätzen Zustands, vgl. Gleichung 2.2 und Gleichung 2.3.

$$\text{Prädiktionsmodell - } p(x_t|u_t, x_{t-1}) \qquad \text{Observationsmodell - } p(z_t|x_t) \tag{2.7}$$

Das Prädiktionsmodell beschreibt hier die Wahrscheinlichkeit eines neuen Zustands x_t unter der Voraussetzung eines Eingangs u_t und des letzten Zustands x_{t-1}. Es spiegelt also wieder, wie wahrscheinlich ein Zustand x_t eintritt, wenn im vorigem Zeitschritt ein bestimmter Zustand x_{t-1} geschätzt wurde und es den Eingang u_t in das System gibt. Das Observationsmodell beschreibt dagegen, wie wahrscheinlich eine Beobachtung z_t im Zustand x_t ist.

2.4 Kalman Filter

2.4.1 Funktionsweise und Algorithmus

Der Kalman Filter ist eine Möglichkeit einen Bayes-Filter zu implementieren. Grundgedanke ist es den Zustand eines Systems als Gauß-Verteilungen zu beschreiben, also den Systemzustand als Erwartungswert $\boldsymbol{\mu_t}$ mit Kovarianz $\boldsymbol{\Sigma_t}$. Über den Bayes-Filter Algorithmus einen neuen Systemzustand auszurechnen, [Kalman, 1960]. Dabei werden die beiden Berechnungsschritte des Bayes Filters als Prädiktion und Observation bezeichnet.

Die Prädiktion, oder auch Vorhersage, ist zu vergleichen mit dem ersten Berechnungsschritt des Bayes-Filter Algorithmus. Hier wird auch aus einer vorherigen Schätzung des Systemzustands $\mathbf{x_{t-1}}$, dem Eingang $\mathbf{u_t}$, und den mathematischen Modellen $\mathbf{B_t}$ und $\mathbf{A_t}$ eine neue Zustandsschätzung $\mathbf{x_t}$ vorgenommen.

$$\mathbf{x_t} = \mathbf{A_t x_{t-1}} + \mathbf{B_t u_t} + \boldsymbol{\varepsilon_t} \tag{2.8}$$

Die Matrix $\mathbf{A_t}$ beschreibt dabei die Zustandsänderung ohne Eingang, also die Eigendynamik des Systems. $\mathbf{B_t}$ stellt einen Zusammenhang zwischen dem Eingang $\mathbf{u_t}$ und dem Zustand $\mathbf{x_t}$ her. Diese Matrizen werden auch als Prädiktionsmodell bezeichnet. Dieses Modell muss linear sein, damit der Systemzustand eine Normalverteilung bleibt (siehe Abbildung 2.6). $\boldsymbol{\varepsilon_t}$ beschreibt einen Rauschterm, welcher die Unsicherheit der Vorhersage modelliert.

Der Observationsschritt gleicht Schritt zwei des Bayes-Filter Algorithmus. Hierzu benötigt man die Matrix $\mathbf{C_t}$, welche die Messung $\mathbf{z_t}$ auf den Zustandsvektor abbildet. Diese Matrix wird wiederum als Observationsmodell bezeichnet. Das Rauschen der Messung wird hier durch die Addition von $\boldsymbol{\delta}$ realisiert.

$$\mathbf{z_t} = \mathbf{C_t}\mathbf{x_t} + \boldsymbol{\delta_t} \tag{2.9}$$

Die einzelnen Parameter haben folgende Dimensionen:

$\mathbf{x_t}$	–	n	*Zustandsvektor*
$\mathbf{A_t}$	–	n x n	*Eigendynamikmatrix*
$\mathbf{B_t}$	–	n x m	*Eingangsmatrix*
$\mathbf{u_t}$	–	m	*Eingangsvektor*
$\boldsymbol{\varepsilon_t}$	–	n x n	*Rauschterm der Vorhersage*
$\mathbf{z_t}$	–	k	*Messvektor*
$\mathbf{C_t}$	–	k x n	*Messmatrix*
$\boldsymbol{\delta_t}$	–	k x n	*Messrauschen*

Tabelle 2.1

Der Algorithmus des Kalman Filters ist in Abbildung 2.3 zu sehen.
Die Eingaben für den Algorithmus sind zum Einen der letzte geschätzte Zustand $\boldsymbol{\mu_{t-1}}$ und die zugehörige Kovarianz $\boldsymbol{\Sigma_{t-1}}$ und zum Anderen der aktuelle Systemeingang $\mathbf{u_t}$ und die aktuelle Messung $\mathbf{z_t}$. Schlussendlich soll ein neuer Erwartungswert $\boldsymbol{\mu_t}$ und seine Kovarianz $\boldsymbol{\Sigma_t}$ berechnet werden. In den Zeilen fünf und sechs wird zunächst

Eingaben

1 $\boldsymbol{\mu_{t-1}}$ // *Erwartungswert des letzten Zeitschritts*
2 $\mathbf{\Sigma_{t-1}}$ // *Kovarianzmatrix des letzten Zeitschritts*
3 $\mathbf{u_t}$ // *Eingang des Systems*
4 $\mathbf{z_t}$ // *Messung*

Algorithmus

5 $\overline{\mu}_t = \mathbf{A_t}\ \mu_{t-1} + \mathbf{B_t u_t}$; // *Prädiktionsschritt des Systemzustands*
6 $\overline{\mathbf{\Sigma}}_\mathbf{t} = \mathbf{A_t}\ \mathbf{\Sigma_{t-1}}\ \mathbf{A_t^T} + \mathbf{R_t}$; // *Prädiktionsschritt der Systemkovarianz*
7 $\mathbf{K_t} = \overline{\mathbf{\Sigma}}_\mathbf{t}\ \mathbf{C_t^T}\ (\mathbf{C_t}\ \overline{\mathbf{\Sigma}}_\mathbf{t}\ \mathbf{C_t^T} + \mathbf{Q_1})^{-1}$; // *Berechnung des Kalman Gain*
8 $\boldsymbol{\mu_t} = \overline{\mu}_\mathbf{t} + \mathbf{K_t}\ (\mathbf{z_t} - \mathbf{C_t}\ \overline{\mu}_\mathbf{t})$; // *Observationsschritt des Systemzustands*
9 $\mathbf{\Sigma} = (\mathbf{I} - \mathbf{K_t}\ \mathbf{C_t})\overline{\mathbf{\Sigma}}_\mathbf{t}$; // *Observationsschritt der Systemkovarianz*

Rückgabe

10 $\boldsymbol{\mu_t}, \mathbf{\Sigma_t}$ // *Neuer Erwartungswert und Kovarianz*

Abbildung 2.3: Kalman Filter Algorithmus

anhand des Systemeingangs $\mathbf{u_t}$ sowohl der neue Systemzustand $\overline{\boldsymbol{\mu_t}}$ als auch die neue Kovarianz $\overline{\mathbf{\Sigma}}_\mathbf{t}$ geschätzt. Dazu wird das Prädiktions Modell mit $\mathbf{A_t}$ und $\mathbf{B_t}$ mit dem Zustand $\boldsymbol{\mu_{t-1}}$ und der Kovarianzmatrix $\mathbf{\Sigma_{t-1}}$ des vorherigen Zeitschrittes verrechnet. Die Matrix $\mathbf{K_t}$ in Zeile sieben wird als Kalman Gain bezeichnet. Sie beschreibt mit welchem Gewicht die Messung $\mathbf{z_t}$ in die endgültige Schätzung $\boldsymbol{\mu_t}$ eingeht. Der Kalman Filter verrechnet über den Kalman Gain $\mathbf{K_t}$ die beiden Erwartungswerte der Prädiktion und Observation nach ihrer Varianz. Die neue Schätzung $\boldsymbol{\mu_t}$ liegt dabei immer näher am Erwartungswert mit der geringeren Varianz. Zu sehen ist dieses Verhalten in Abbildung 2.4.
In den Zeilen acht und neun werden dann $\boldsymbol{\mu_t}$ und $\mathbf{\Sigma_t}$ aus den Schätzungen $\overline{\boldsymbol{\mu_t}}$, $\overline{\mathbf{\Sigma}}_\mathbf{t}$, der Messung $\mathbf{z_t}$, dem Observationsmodell $\mathbf{C_t}$ und dem Kalman Gain $\mathbf{K_t}$ berechnet. In Abbildung 2.4 sieht man den Ablauf des Kalman Algorithmus für ein 1-D System. Während des ersten Durchlaufs war die Messung genauer als die erste Schätzung und

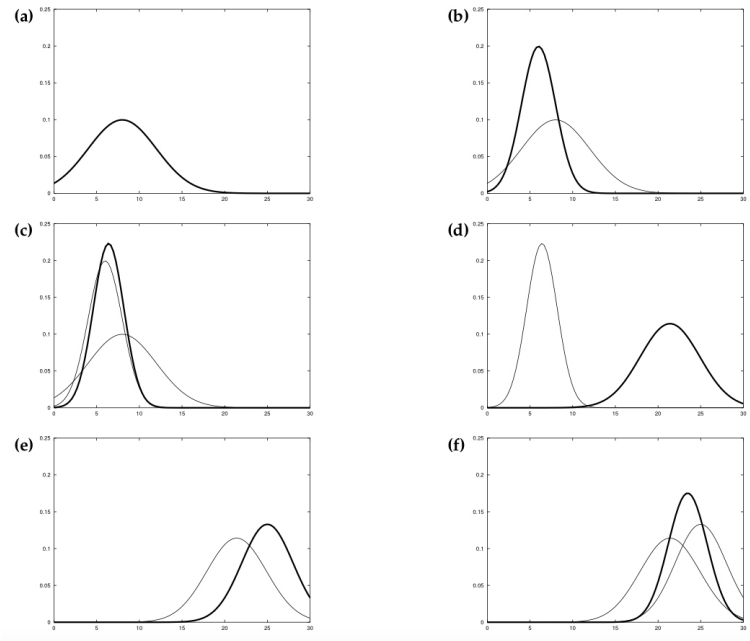

Abbildung 2.4: Kalman Filter Algorithmus für ein 1-D System, Quelle: [Thrun et al., 2006]
In Bild (a) ist die erste Vorhersage für den nächsten Zeitschritt zu erkennen und in (b) die dazugehörige Messung. Aus diesen beiden Gauß-Kurven wird dann der neue Zustand in (c) geschätzt. Danach folgt wieder eine Prädiktion des neuen Zustandes (d) und eine Messung (e). In (f) werden diese wieder verrechnet und ein neuer Systemzustand gebildet.

der neue Erwartungswert liegt nahe am Erwartungswert der Messung. Im zweiten Durchgang sind Messung und Schätzung ähnlich ungenau und der neue Erwartungswert liegt somit zwischen dem geschätzten und gemessenen Systemzustand. Außerdem ist zu erkennen, dass die Varianz des neuen Zustands immer geringer ist als die der Messung und der Schätzung. Man nimmt also an, dass Schätzung und Messung zusammen ein genaueres Ergebnis liefern als alleine.

Das Interessante beim Kalman Filter ist, dass er auch ohne Observation funktioniert, wie es in Abbildung 2.15 zu sehen ist. Sollte bei der Berechnung für einen Zeitschritt keine Messung vorhanden sein, so liefert der Algorithmus trotzdem einen neuen Zu-

standsvektor, da dieser schon im Prädiktionsschritt gebildet wird, siehe Gleichung 2.8. Jedoch wird $\mathbf{x_t}$ durch die Addition von $\boldsymbol{\varepsilon_t}$ mit jedem Zeitschritt ungenauer.

2.4.2 Modelle

Um ein System mathematisch beschreiben zu können, benötigt man ein Modell. Dieses Modell spiegelt Eigenschaften des Systems wieder und beschreibt sein Verhalten. Beim Kalman Filter besteht solch ein Modell aus 3 Teilen, [Thrun et al., 2006].

- $\mathbf{x_t}$ – Systemzustand
- $\mathbf{A_t}$ – Eigendynamik
- $\mathbf{B_t}$ – Systemeingangsmodell

Der Systemzustand $\mathbf{x_t}$ muss sich dabei nicht auf beobachtbare Eigenschaften beschränken. Durch den Algorithmus des Kalman Filters, siehe Abbildung 2.3, ist es möglich auch nicht beobachtbare Eigenschaften zu beschreiben. Durch Verrechnung von $\mathbf{A_t}$ und der Kovarianz $\mathbf{\Sigma_t}$ (Zeile 6 in Abbildung 2.3) und den Zusammenhängen, die in $\mathbf{A_t}$ beschrieben sind, kann man über die Eigendynamik nicht beobachtbare Eigenschaften filtern, siehe Abbildung 2.5. Dadurch ist es möglich mit einem Kalman Filter verschiedenste Systeme und ihre Eigendynamik zu beschreiben, auch wenn man dafür Parameter braucht, die nicht gemessen werden können.

Bedingung für das Funktionieren des Kalman Filter Algorithmus ist, dass sowohl das Prädiktions- als auch das Observationsmodell lineare Modelle sind. Das ist notwendig, damit der Zustand des Systems auch nach den Berechnungen des Filters normalverteilt bleibt, wie in Abbildung 2.6 gezeigt wird. Da jedoch die meisten Systeme nichtlinear sind, braucht man eine Alternative zum normalen Kalman Filter. Zwei davon werden in den nächsten beiden Abschnitten beschrieben. Zum Einen der Extended Kalman Filter und zum Anderen der Unscented Kalman Filter.

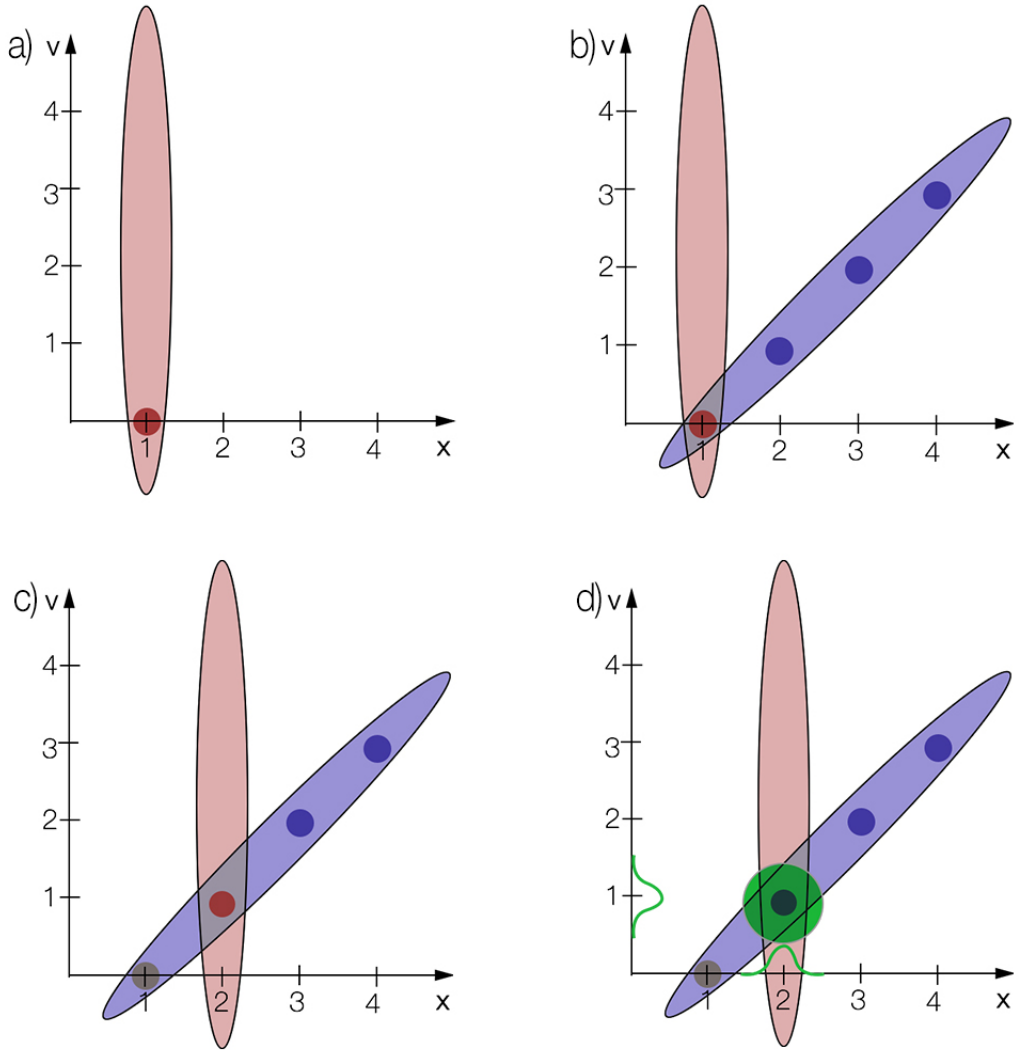

Abbildung 2.5: Filtern von unbekannten Eigenschaften

In Bild a) ist das System an Position $x = 1$ und es gibt keinerlei Informationen über die Geschwindigkeit. In b) ist die Systembeschreibung $x_t(t) = x_{t-1} + vt$ als 2-D Gauß-Kurve dargestellt. Diese würde alleine aber auch keine Aussage über den Ort oder die Geschwindigkeit des Systems treffen. In c) gibt es eine Messung nach einem Zeitschritt an Position $x = 2$. Durch die Multiplikation von den beiden Gauß-Kurven kann man in d) eine Aussage über die Geschwindigkeit machen, obwohl diese nicht gemessen wurde.

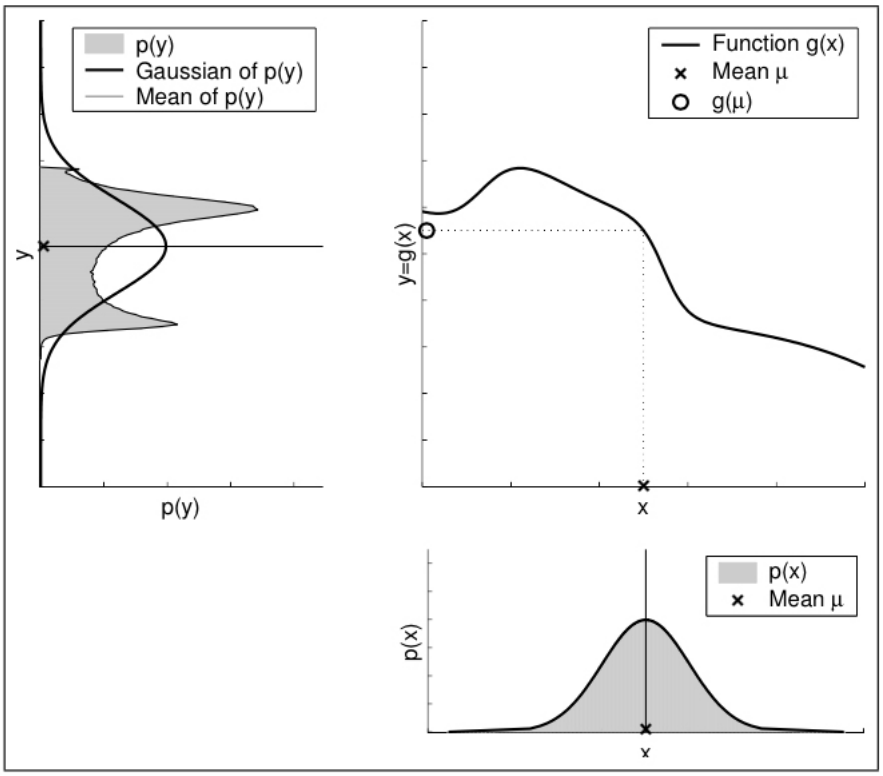

Abbildung 2.6: Kalman Filter bei nichtlinearen Funktionen, Quelle: [Thrun et al., 2006]

Wird die Normalverteilung (unten rechts) mit der nichtlinearen Funktion $g(x)$ (oben rechts) multipliziert, so erhält man die graue Fläche $p(y)$ (oben links). Wie man sieht, ist dies keine Normalverteilung mehr. Die eigentliche Normalverteilung von $p(y)$ ist hier mit der schwarzen Linie im Bild links oben gekennzeichnet. Diese soll mit Hilfe des Extended- und Unscented Kalman Filters approximiert werden.

2.5 Extended Kalman Filter

Der Extended Kalman Filter bietet die Möglichkeit auch nicht lineare Prädiktions- und Observationsmodelle mit dem Kalman Filter Algorithmus zu verwenden, [Welch und Bishop, 1995] [Jul, 1997]. Dazu werden die Modellfunktionen durch eine Taylor Reihe linearisiert, [Thrun et al., 2006, Seite 56 ff.].

$$\mathbf{g'}(\mathbf{u_t}, \mathbf{x_{t-1}}) := \frac{\partial\ \mathbf{g}(\mathbf{u_t}, \mathbf{x_{t-1}})}{\partial\ \mathbf{x_{t-1}}} \tag{2.10}$$

Dabei ist es wichtig die richtige Stützstelle für die Linearisierung zu wählen. Beim Prädiktionsschritt bietet es sich an, den Erwartungswert der letzten Schätzung $\boldsymbol{\mu_{t-1}}$ zu verwenden, da dieser der wahrscheinlichste Systemzustand ist.

$$\begin{aligned} g(\mathbf{u_t}, \mathbf{x_{t-1}}) &\approx g(\mathbf{u_t}, \boldsymbol{\mu_{t-1}}) + \underbrace{g'(\mathbf{u_t}, \boldsymbol{\mu_{t-1}})}_{=:\ \mathbf{G_t}} (\mathbf{x_{t-1}} - \boldsymbol{\mu_{t-1}}) \\ &= g(\mathbf{u_t}, \boldsymbol{\mu_{t-1}}) + \mathbf{G_t}(\mathbf{x_{t-1}} - \boldsymbol{\mu_{t-1}}) \end{aligned} \tag{2.11}$$

Analog dazu wird beim Observationsmodell der neue geschätzte Systemzustand $\overline{\boldsymbol{\mu}}_{\boldsymbol{t}}$ als Linearisierungspunkt verwendet.

$$\begin{aligned} h(\mathbf{x_t}) &\approx h(\overline{\boldsymbol{\mu}}_{\boldsymbol{t}}) + \underbrace{h'(\overline{\boldsymbol{\mu}}_{\boldsymbol{t}})}_{=:\ \mathbf{H_t}} (\mathbf{x_t} - \overline{\boldsymbol{\mu}}_{\boldsymbol{t}}) \\ &= h(\overline{\boldsymbol{\mu}}_{\boldsymbol{t}}) + \mathbf{H_t}(\mathbf{x_t} - \overline{\boldsymbol{\mu}}_{\boldsymbol{t}}) \end{aligned} \tag{2.12}$$

Bei einem mehrdimensionalen System werden die Ableitungen $g'(\mathbf{u_t}, \mathbf{x_{t-1}})$ und $h'(\overline{\boldsymbol{\mu}}_{\boldsymbol{t}})$ durch die Jacobimatritzen $\mathbf{G_t}$ und $\mathbf{H_t}$ realisiert.

Wie man in Abbildung 2.7 sieht, ist der Algorithmus sehr ähnlich zum normalen Kalman Filter Algorithmus aus Abbildung 2.3. In Zeile fünf wird jedoch der neue Zustand nicht mehr über die Matritzen $\mathbf{A_t}$ und $\mathbf{B_t}$ geschätzt, sondern durch die nichtlineare Funktion $g(\mathbf{u_t}, \boldsymbol{\mu_{t-1}})$. Genauso bei der Berechnung von $\boldsymbol{\mu_t}$, bei der jetzt $h(\overline{\boldsymbol{\mu}}_{\boldsymbol{t}})$ verwendet wird. Für die neue Kovarianzmatrix und den Kalman Gain werden die linearisierten Modellmatritzen $\mathbf{G_t}$ und $\mathbf{H_t}$ benutzt. Somit ist der Systemzustand auch

Eingaben

1 $\boldsymbol{\mu_{t-1}}$ // *Erwartungswert des letzten Zeitschritts*
2 $\boldsymbol{\Sigma_{t-1}}$ // *Kovarianzmatrix des letzten Zeitschritts*
3 $\mathbf{u_t}$ // *Eingang des Systems*
4 $\mathbf{z_t}$ // *Messung*

Algorithmus

5 $\overline{\mu}_\mathbf{t} = g(\mathbf{u_t}, \boldsymbol{\mu_{t-1}})$; // *Prädiktion des neuen Zustands über die Funktion g*
6 $\overline{\boldsymbol{\Sigma}}_\mathbf{t} = \mathbf{G_t}\ \boldsymbol{\Sigma_{t-1}}\ \mathbf{G_t^T} + \mathbf{R_t}$; // *Prädiktionsschritt der Kovarianz des System*
7 $\mathbf{K_t} = \overline{\boldsymbol{\Sigma}}_\mathbf{t}\ \mathbf{H_t^T}\ (\mathbf{H_t}\ \overline{\boldsymbol{\Sigma}}_\mathbf{t}\ \mathbf{H_t^T} + \mathbf{Q_1})^{-1}$; // *Berechnung des Kalman Gain*
8 $\boldsymbol{\mu_t} = \overline{\mu}_\mathbf{t} + \mathbf{K_t}(\mathbf{z_t} - h(\overline{\boldsymbol{\mu}}_\mathbf{t}))$; // *Observationsschritt des Systemzustands*
9 $\boldsymbol{\Sigma} = (\mathbf{I} - \mathbf{K_t}\ \mathbf{H_t})\ \overline{\boldsymbol{\Sigma}}_\mathbf{t}$; // *Observationsschritt der Kovarianzmatrix*

Rückgabe

10 $\boldsymbol{\mu_t}, \boldsymbol{\Sigma_t}$ // *Neuer Eigenwert und Kovarianz*

Abbildung 2.7: Extended Kalman Filter Algorithmus

nach der Verwendung von nichtlinearen Modellen normalverteilt. In Abbildung 2.8 ist die Linearisierung nocheinmal dargestellt.

Der Vorteil dieser Art der Linearisierung liegt vor allem in der Performance der Berechnung. Der Nachteil ist der auftretende Linearisierungsfehler, der je nach Modell und Erwartungswert unterschiedlich groß ist, siehe Abbildung 2.9.

2.6 Unscented Kalman Filter

Eine weitere Möglichkeit um nichtlineare Modelle zu verwenden ist der Unscented Kalman Filter [Julier und Uhlmann, 2004]. Die Idee ist ähnlich wie die des Partikelfilters in Abschnitt 2.7. Der Algorithmus des Unscented Kalman Filter ist darauf ausgelegt eine Gauß-Funktion zu approximieren, im Gegensatz zum Partikelfilter, welcher eine beliebige Funktion approximieren kann.

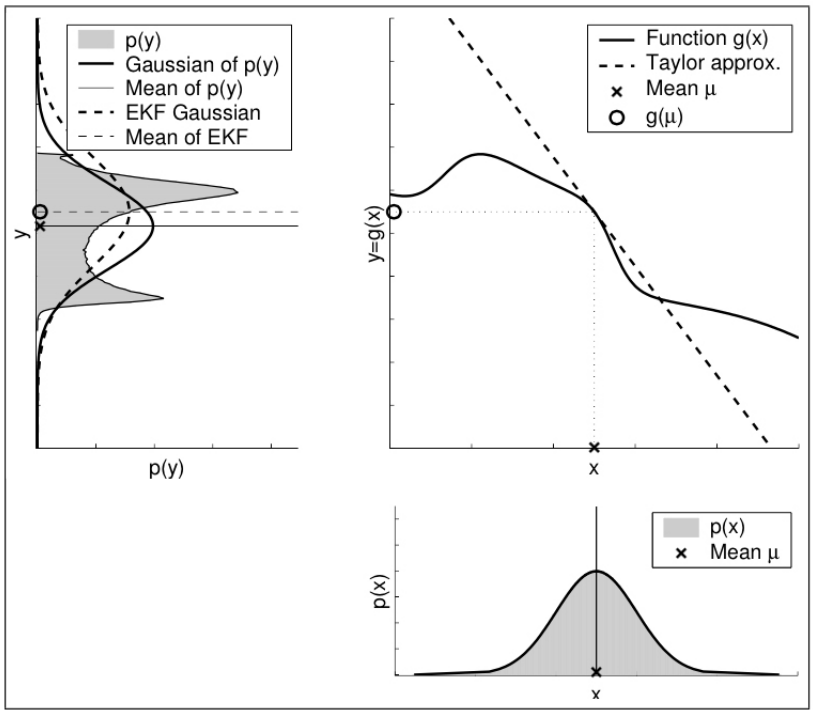

Abbildung 2.8: Linearisierung und Anwendung des Extended Kalman Filter Algorithmus, Quelle: [THRUN et al., 2006]
In dieser Abbildung ist zu sehen, wie die nichtlineare Funktion $g(x)$ an der Stelle μ_{t-1} linearisiert wird um wieder eine Gauß-Funktion zu erhalten. Außerdem ist auch der Fehler zu erkennen, den der EKF Algorithmus durch die Linearisierung mit sich bringt.

Die Linearisierung erfolgt über sogenannte Sigma Punkte $\mathbf{X}^{[i]}$. Diese haben den Sinn die nichtlineare Funktion über die gesamte Breite der Normalverteilung zu approximieren.

Dazu werden $2n + 1$ Sigma Punkte entlang der Gauß-Kurve gelegt. Der erste Sigma Punkt ist der Erwartungswert der Gauß-Verteilung gefolgt von $2n$ Punkten symmetrisch vom Erwartungswert ausgehend. Diese werden dann in die nichtlineare Funktion

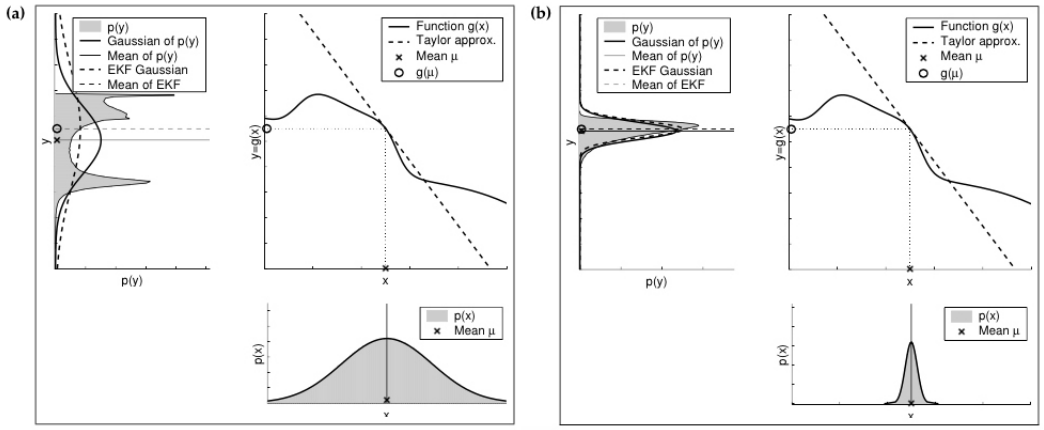

Abbildung 2.9: Abweichungsfehler des Extended Kalman Filter, Quelle: [Thrun et al., 2006]

Im linken Bild a) ist zu erkennen, dass die Approximation durch die Linearisierung bei einer hohen Varianz einen großen Fehler erzeugt und ein ungenaues Ergebnis zur Folge hat. Bei einer geringen Varianz wie in Bild b) ist der Fehler deutlich geringer, da die Linearisierung die nichtlineare Funktion lokal gut approximiert.

gegeben und später aus diesen Punkten dann wieder einer Normalverteilung gebildet.

$$\begin{aligned} \mathbf{X}^{[0]} &= \boldsymbol{\mu} \\ \mathbf{X}^{[i]} &= \boldsymbol{\mu} + \left(\sqrt{(n+\lambda)\boldsymbol{\Sigma}}\right)_i \qquad \text{for } i = 1, \ldots, n \\ \mathbf{X}^{[i]} &= \boldsymbol{\mu} - \left(\sqrt{(n+\lambda)\boldsymbol{\Sigma}}\right)_{i-n} \qquad \text{for } i = n+1, \ldots, 2n \end{aligned} \tag{2.13}$$

Hierbei ist $\lambda = \alpha^2(n+\kappa) - n$. Die Entfernung der Punkte zum Erwartungswert bestimmen also α und κ. Um die Lage der Sigmapunkte auf der Gauß-Kurve zu beschreiben, gibt es zu jeden Sigmapunkt noch 2 Gewichte, $w_m^{[i]}$ und $w_c^{[i]}$. Das erste Gewicht $w_m^{[i]}$ dient der Berechnung des neuen Erwartungswertes und das zweite Gewicht $w_c^{[i]}$ dient der Berechnung der Kovarianz der neuen Normalverteilung.

$$\begin{aligned} w_m^{[o]} &= \tfrac{\lambda}{n+\lambda} \\ w_c^{[o]} &= \tfrac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \\ w_m^{[i]} &= w_c^{[i]} = \tfrac{1}{2(n+\lambda)} \qquad \text{for } i = 1, \ldots, 2n \end{aligned} \tag{2.14}$$

Alle Sigmapunkte werden nun mit der nichtlinearen Funktion $g(\mathbf{X})$ verrechnet.

$$\mathbf{Y}^{[i]} = g(\mathbf{X}^{[i]}) \tag{2.15}$$

Der neue Erwartungswert $\boldsymbol{\mu'}$ und die neue Kovarianz $\boldsymbol{\Sigma'}$ werden dann aus den Gewichten $w_m^{[i]}$ und $w_c^{[i]}$ zusammen mit den Funktionswerten $\mathbf{Y}^{[i]}$ berechnet.

$$\begin{aligned} \boldsymbol{\mu'} &= \textstyle\sum_{i=0}^{2n} w_m^{[i]} \mathbf{Y}^{[i]} \\ \boldsymbol{\Sigma'} &= \textstyle\sum_{i=0}^{2n} w_c^{[i]} \left(\mathbf{Y}^{[i]} - \boldsymbol{\mu'}\right)^T \end{aligned} \tag{2.16}$$

In Abbildung 2.10 ist der Algorithmus noch einmal detailliert aufgeführt und in Abbildung 2.11 bildlich dargestellt.

Der Algorithmus des Unscented Kalman Filters ist durch die Berechnung der Sigmapunkte und Gewichte, zusammen mit der nachträglichen Wiederherstellung der Gauß-Form des Systemzustands, komplexer als der Algorithmus des Extended Kalman Filters.
Der Unscented Kalman Filter hat im Gegensatz zum Extended Kalman Filter zwei große Vorteile. Auf der einen Seite werden für die Berechnung keine Ableitungen der Modellfunktionen benötigt. Auf der anderen Seite ist der Approximationsfehler beim UKF geringer als beim EKF, da der UKF noch bis zum zweiten Taylorglied genau arbeitet, während der EKF nur im ersten Taylorglied ein genaueres Ergebnis liefert. Dies wird besonders bei einer großen Varianz deutlich, siehe Abbildung 2.12. Durch die Sigma Punkte und ihre Gewichte approximiert der Unscented Kalman Filter auch bei einer hohen Varianz noch gut.

Eingaben

1 μ_{t-1} // *Erwartungswert des letzten Zeitschritts*

2 $\mathbf{\Sigma}_{t-1}$ // *Kovarianzmatrix des letzten Zeitschritts*

3 $\mathbf{u}_t$ // *Eingang des Systems*

4 $\mathbf{z}_t$ // *Messung*

Algorithmus

5 $\mathbf{X}_{t-1} = (\boldsymbol{\mu_{t-1}} \quad \boldsymbol{\mu_{t-1}} + \boldsymbol{\gamma}\sqrt{\boldsymbol{\Sigma_{t-1}}} \quad \boldsymbol{\mu_{t-1}} - \boldsymbol{\gamma}\sqrt{\Sigma_{t-1}})$; // *Berechnung der Sigmapunkte*

6 $\overline{\mathbf{X}}_t^* = g(\boldsymbol{u_t}, \mathbf{X}_{t-1})$; // *Prädiktionsschritt mit Transformation der Sigmapunkte über die Funktion g*

7 $\overline{\boldsymbol{\mu}}_{\boldsymbol{t}} = \sum_{i=0}^{2n} w_m^{[i]} \mathbf{X}_t^{*[i]}$; // *Berechnung des prädizierten Systemzustands*

8 $\overline{\boldsymbol{\Sigma}} = \sum_{i=0}^{2n} w_c^{[i]} (\mathbf{X}_t^{*[i]} - \overline{\boldsymbol{\mu}}_{\boldsymbol{t}})(\mathbf{X}_t^{*[i]} - \overline{\boldsymbol{\mu}}_{\boldsymbol{t}})^T + \boldsymbol{R_t}$; // *Berechnung der prädizierten Kovarianzmatrix*

9 $\overline{\mathbf{X}}_{\mathbf{t}} = (\overline{\boldsymbol{\mu}}_{\boldsymbol{t}} \quad \overline{\boldsymbol{\mu}}_{\boldsymbol{t}} + \boldsymbol{\gamma}\sqrt{\overline{\boldsymbol{\Sigma}}_{\boldsymbol{t}}} \quad \overline{\boldsymbol{\mu}}_{\boldsymbol{t}} - \boldsymbol{\gamma}\sqrt{\overline{\boldsymbol{\Sigma}}_{\boldsymbol{t}}})$; // *Berechnung der Sigmapunkte aus der Prädiktion*

10 $\overline{\mathbf{Z}}_t = h(\overline{\mathbf{X}}_t)$; // *Transformation der Prädiktions zur geschätzen Messung*

11 $\hat{\boldsymbol{z}}_{\boldsymbol{t}} = \sum_{i=0}^{2n} w_m^{[i]} \overline{\mathbf{Z}}_t^{[i]}$; // *Berechnung der Prädizierten Messung*

12 $\boldsymbol{S_t} = \sum_{i=0}^{2n} w_c^{[i]} (\overline{\mathbf{Z}}_t^{[i]} - \hat{\boldsymbol{z}}_{\boldsymbol{t}})(\overline{\mathbf{Z}}_t^{[i]} - \hat{\boldsymbol{z}}_{\boldsymbol{t}})^T + \boldsymbol{Q_t}$; // *Berechnung der prädizierten Kovarianzmatrix der Messung*

13 $\overline{\boldsymbol{\Sigma}}_{\boldsymbol{t}}^{\boldsymbol{x},\boldsymbol{z}} = \sum_{i=0}^{2n} w_c^{[i]} (\overline{\mathbf{X}}_t^{[i]} - \overline{\boldsymbol{\mu}}_{\boldsymbol{t}})(\overline{\mathbf{Z}}_t^{[i]} - \hat{\boldsymbol{z}}_{\boldsymbol{t}})^T$; // *Berechnung der Kovarianz zwischen der Prädiktion und der prädizierten Messung*

14 $\boldsymbol{K_t} = \overline{\boldsymbol{\Sigma}}_{\boldsymbol{t}}^{\boldsymbol{x},\boldsymbol{z}} {\boldsymbol{S_t}}^{-1}$; // *Berechnung des Kalman Gain*

15 $\boldsymbol{\mu_t} = \overline{\boldsymbol{\mu}}_{\boldsymbol{t}} + \boldsymbol{K_t}(\boldsymbol{z_t} - \hat{\boldsymbol{z}}_{\boldsymbol{t}})$; // *Observationsschritt des Systemzustands*

16 $\boldsymbol{\Sigma_t} = \overline{\boldsymbol{\Sigma}}_{\boldsymbol{t}} - \boldsymbol{K_t S_t} {\boldsymbol{K_t}}^T$; // *Observationsschritt der Kovarianzmatrix des Systems*

Rückgabe

17 $\boldsymbol{\mu_t}, \boldsymbol{\Sigma_t}$ // *Neuer Erwartungswert und Kovarianz*

Abbildung 2.10: Unscented Kalman FilterAlgorithmus

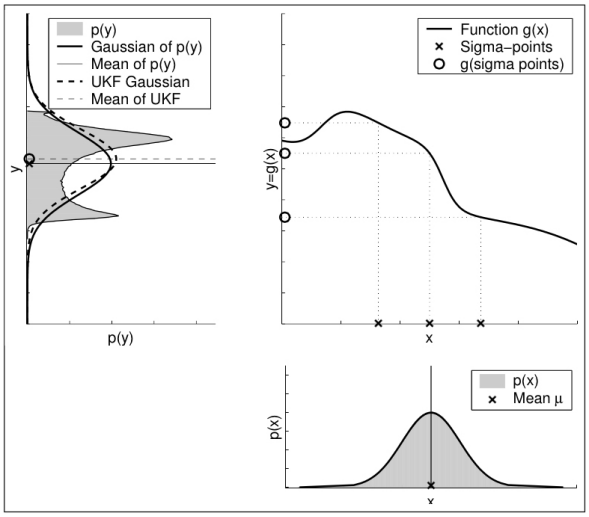

Abbildung 2.11: Unscented Kalman Filter, Quelle: [Thrun et al., 2006]
In diesem Beispiel ist $n = 1$, da 3 Sigma Punkte verwendet werden um die Normalverteilung mit der nichtlinearen Funktion zu multiplizieren. Die gestrichelte Linie im linken Bild ist das Ergebnis nach der Berechnung von $\boldsymbol{\mu}'$ und $\boldsymbol{\Sigma}'$.

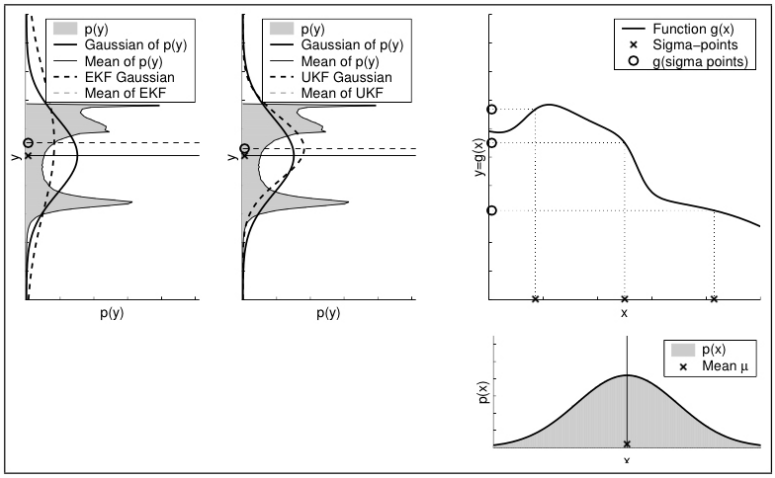

Abbildung 2.12: Vergleich zwischen Unscented und Extended Kalman Filter, Quelle: [Thrun et al., 2006]
Selbst bei einer großen Varianz zeigt der UKF immer noch eine gute Performance (mittleres Bild) im Gegensatz zum EKF (linkes Bild). Der Unterschied zwischen der Gauß-Verteilung von p(y) (durchgezogene Linie) und der Approximation durch den UKF ist sehr viel geringer als beim EKF.

2.7 Partikelfilter

Der Partikelfilter ist eine weitere Art um den Bayes Algorithmus zu implementieren. Ein Partikel repräsentiert immer einen möglichen Zustand des Systems. Ein Partikelfilter besteht dabei immer aus einem Set X_t von Partikeln, [Thrun et al., 2006, Seite 96 ff.].

$$X_t := \mathbf{x}_t^{[1]}, \mathbf{x}_t^{[2]}, \ldots, \mathbf{x}_t^{[M]} \tag{2.17}$$

M ist oft eine große Zahl, z.b. $M = 1000$. Jedes Partikel $\mathbf{x}_t^{[m]} mit : \ 1 \leq m \leq M$ ist also ein möglicher Systemzustand zum Zeitpunkt t. Der Partikelfilter kann durch diese Partikel verschiedene Verteilungen repräsentieren und nichtlineare Systeme gut modellieren, siehe Abbildung 2.13.
Der Algorithmus ist in Abbildung 2.14 zu sehen. Die Eingänge des Algorithmus sind das alte Partikel-Set X_t, der Systemeingang $\mathbf{u}_t$ und die Messung des Systems $\mathbf{z}_t$. Wie auch beim Kalman Filter arbeitet der Partikelfilter rekursiv über die Zeit. Das Partikel-Set X_t wird also aus dem Partikel-Set X_{t-1} gebildet. Zunächst wird für jedes Partikel eine neuer Zustand über den Eingang u_t ausgerechnet, siehe Zeile sechs des Algorithmus.

$$x_t^{[m]} \ \sim \ p(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x}_{t-1}^{[m]}) \quad for \ m = 1, \ldots, M \tag{2.18}$$

Anschließend wird in Zeile sieben für jedes der Partikel ein Gewicht $w_t^{[m]}$ ausgerechnet, welches die Schätzung $\mathbf{x}_t^{[m]}$ mit der Messung $\mathbf{z}_t$ vergleicht. Dieses Gewicht beschreibt wie ähnlich sich das neue Partikel $\mathbf{x}_t^{[m]}$ und die Messung $\mathbf{z}_t$ sind. Je ähnlicher sich Messung und Partikel sind, desto "wichtiger" ist dieses Partikel.

$$w_t^{[m]} \ = \ p(\mathbf{z}_t \mid \mathbf{x}_t^{[m]}) \tag{2.19}$$

Das neue Partikel-Set und seine Gewichte werden dann zu einem Vektor $\langle \mathbf{x}_t^{[m]}, w_t^{[m]} \rangle$ zusammengefasst und das Set $\overline{X}_t$ in Zeile acht gebildet. Der nächste und für den Partikelfilter entscheidende Schritt beruht auf dem darwinistischen Prinzip "survival

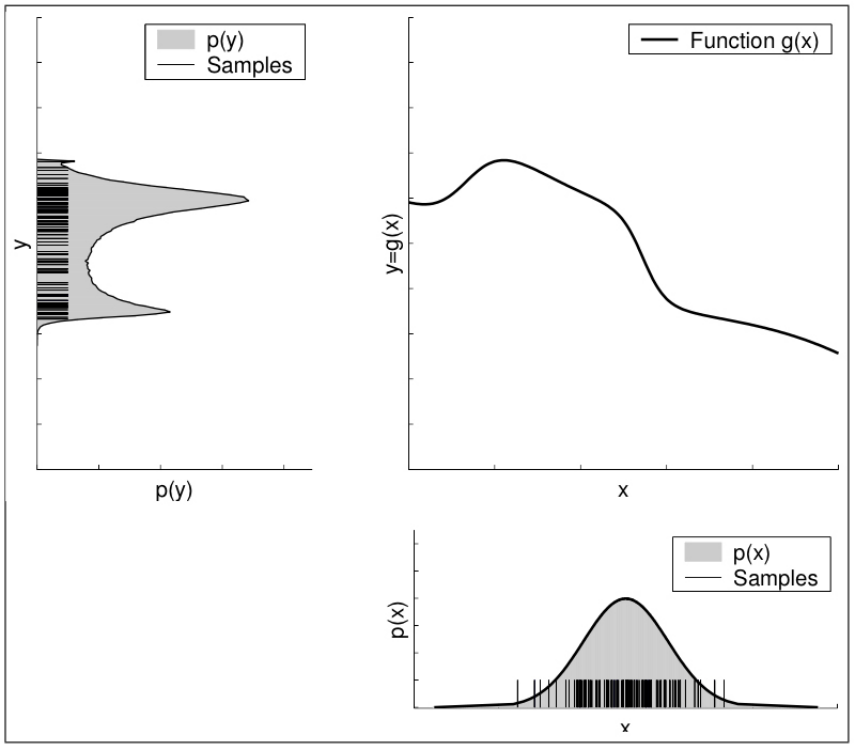

Abbildung 2.13: Partikelfilter, Quelle: [THRUN et al., 2006]
Das Partikel-Set (Bild unten rechts) wird durch die Funktion g(x) (Bild oben rechts) transformiert. Dadurch ergibt sich eine neue Verteilung der Partikel (Bild oben links). Durch die Transformation mit einer nichtlinearen Funktion löst sich die Gauß-Verteilung des Anfangsets auf.

of the fittest", welche in den Zeilen 10 bis 13 zu finden ist. Für das neue Partikel Set X_t werden zufällig Partikel aus dem Set $\overline{X}_t$ gewählt. Partikel mit einem höheren Gewicht haben dabei eine höhere Wahrscheinlichkeit gewählt zu werden. Das hat zur Folge, dass Partikel mit einem höheren Gewicht $w_t^{[m]}$ öfter ausgewählt werden und mehrmals im neuen Set vorkommen. Partikel mit einem niedrigeren Gewicht haben eine höhere Wahrscheinlichkeit nicht ausgewählt zu werden und "sterben".

Eingaben

1 X_{t-1} // *Partikel Set des letzten Zeitschritts*
2 $\mathbf{u}_t$ // *Eingang des Systems*
3 $\mathbf{z}_t$ // *Messung*

Algorithmus

4 $\overline{X}_t = X_t = \emptyset$; // *Initialisieren der neuen noch leeren Sets.*
5 *for* $m = 1$ *to* M *do*; // *über die komplette Länge des Sets*
6 *sample* $\mathbf{x}_t^{[m]} \sim p(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x}_{t-1}^{[m]})$; // *Schätzung des neuen Zustands eines Partikels*
7 $w_t^{[m]} = p(\mathbf{z}_t \mid \mathbf{x}_t^{[m]})$; // *Berechnung des Gewichts der Schätzung*
8 $\overline{X}_t = \overline{X}_t + \langle \mathbf{x}_t^{[m]}, w_t^{[m]} \rangle$; // *Einfügen des Partikel-Gewicht-Vektors in das Schätzungsset*
9 *endfor*;
10 *for* $m = 1$ *to* M *do*; // *über die komplette Länge des Sets*
11 *draw* i *with probability* $\propto w_t^{[i]}$; // *Auswahl eines Gewichts über seine Wahrscheinlichkeit*
12 *add* $\mathbf{x}_t^{[i]}$ *to* X_t; // *Hinzufügen des Partikels zum ausgewählten Gewicht*
13 *endfor*;

Rückgabe

14 X_t // *Neues Partikelset*

Abbildung 2.14: Partikelfilter Algorithmus

2.8 Tracking

Das Tracking beschreibt das Erkennen und Verfolgen von Objekten, [Volkhardt et al., 2013]. Für diese Arbeit steht vor allem das Personentracking im Vordergrund und die Techniken zum Verfolgen dieser Personen. Verfolgen heißt dabei die Position einer erkannten Person über einen längeren Zeitraum zu schätzen. Dies soll auch geschehen, wenn die Person eine kurze Zeit nicht vom Roboter erkannt wurde, wie es in Abbildung 2.15 dargestellt ist.

Diese Aufgabe lösen die oben vorgestellten Kalman Filter. Jede Person wird im Tracker also als normalverteilte Hypothese mit einem Erwartungswert $\boldsymbol{\mu}_t$ und einer Kovarianz $\boldsymbol{\Sigma}_t$ angenommen. Durch das Prädiktionsmodell kann diese Hypothese zum nächsten Zeitschritt geschätzt werden. Der Personendetektor stellt anschließend eine Messung bereit. Durch den Kalman Filter können außerdem Eigenschaften der Person, die über die Personenerkennung nicht messbar sind, wie z.B. ihre Geschwindigkeit, geschätzt werden.

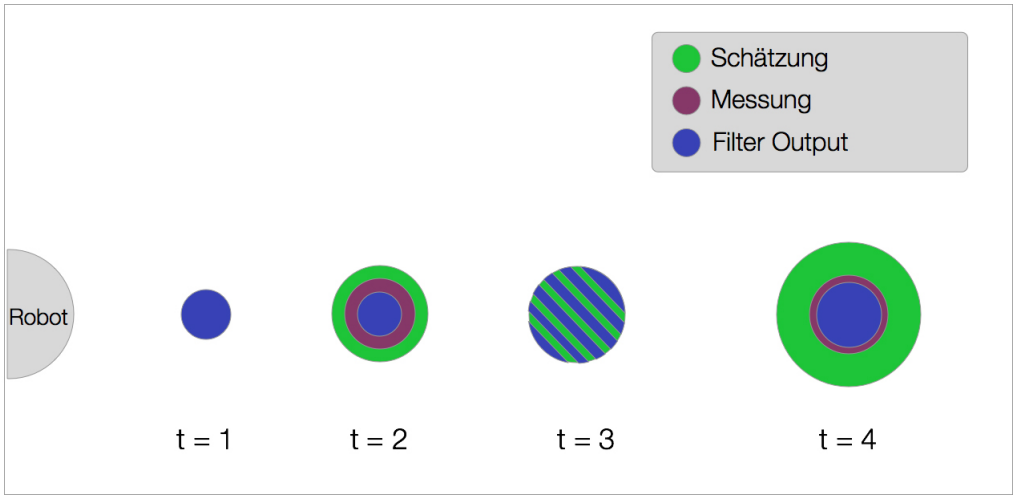

Abbildung 2.15: Tracking Beispiel

Eine Person bewegt sich vom Roboter (links) weg. Dabei liefert der Kalman Algorithmus Schätzungen (grün) , die durch Messungen (braun) zu einem Filter Output (blau) führen. In Zeitschritt $t = 3$ gibt es keine Messung, aber trotzdem einen Filter Output, da der Kalman Filter auch ohne eine Messung einen neuen Zustand liefert. Die Unsicherheit der Hypothese ist durch den Durchmesser der Kreise gekennzeichnet. Je größer der Kreis, desto unsicherer ist die Hypothese.

Kapitel 3

State of the Art

3.1 Personentracking

Im Bereich der mobilen Robotik ist die Thematik des Personentrackings ein wichtiges und gut erforschtes Thema. Für diese Arbeit wird als Grundlage das Tracking System, welches in [Volkhardt et al., 2013] beschrieben wird, verwendet.
Grundlage des Trackings ist immer die Erkennung von Objekten, bzw. Personen durch den Roboter. Da es in diesem Gebiet sehr viele verschiedene Ansätze gibt und die Erkennung kein Teil dieser Arbeit ist, werden im Folgenden nur kurz die Verfahren vorgestellt, die in [Volkhardt et al., 2013] verwendet werden. Aufgabe des Trackers ist es dann diese Verfahren zu fusionieren und in ein gemeinsames Koordinatensystem zu transformieren.

3.1.1 Personenerkennung

Ziel der Personenerkennung ist es für den Tracker eine Personenhypothese bereitzustellen. Das Erkennen von Personen soll dabei auch bei unterschiedlichen Umweltbedingungen, z.B. variierende Lichtverhältnisse, teilweise Verdeckung der Person und unterschiedliche Ansichten der Person, gut funktionieren. Daher werden in [Volkhardt et al., 2013] verschiedene Erkennungsmethodiken genutzt, um ein robustes Erkennungssystem zu realisieren.

- **HOG-Detektor** [Dalal und Triggs, 2005]: Es wird ein *Histograms of Oriented Gradients* (HOG) Detektor angewandt. Im Hintergrund dieses Detektors arbeitet ein Kanten Klassifikator. Somit werden Personen anhand ihrer inneren und äußeren Strukturen erkannt, wie Kleidung oder Körperform.

- **Gesichts-Detektor**: Der Detektor erkennt Gesichter im Kamerabild des Roboters. Aus Performancegründen wird nur der obere Teil des Bildes verarbeitet. Für die Erkennung wird der AdaBoost Detektor von Viola und Jones [Viola und Jones, 2004] benutzt.

- **Bewegungs-Erkennung**: Es werden 2 aufeinanderfolgende Bilder verglichen um Bewegungen in der Nähe des Roboters zu erkennen. Dies ist nur möglich, wenn sich der Roboter selber nicht bewegt.

- **Bein-Detektor**: Über den Distanzlaser des Roboters können Beinpaare erkannt werden [Arras et al., 2007]. Ein Klassifikator trennt Beinpaare von anderen beinähnlichen Objekten, z.B. Tischbeinen.

- **Fastest Pedestrian Detector in the West (FPDW)** [Dollar et al., 2010]: Der FPDW nutz eine Kombination verschiedener Features, um eine schnelle Personenerkennung zu ermöglichen.

- **Part-HOG** [Felzenszwalb et al., 2010]: Der Part HOG nutzt die selben Grundlagen wir der HOG, jedoch gibt es hier transformierbare, flexible Klassifikationsfenster, die auf das Bild angewendet werden. Damit können Personen auch in verschiedenen Posen und unter teilweiser Verdeckung erkannt werden.

Die einzelnen Ergebnisse der Detektoren werden dann zusammengeführt und zu einer Personenhypothese verarbeitet. Dabei werden zunächst die einzelnen Detektionen in Weltkoordinaten transformiert und dann die Positionen als Gauß-Verteilungen angenommen. Der Erwartungswert der Gauß-Verteilung ist die geschätzte Kopfposition der Person. Da Kamerabilder ungenauer bei der Entfernungsschätzung sind, erhalten diese eine höhere Varianz in der Tiefe als die Detektion des Bein-Detektors. Schlussendlich

werden alle Normalverteilungen zu einer Hypothese zusammengeführt. Diese Schritte sind in Abbildung 3.1 auf der linken Seite aufgeführt.

3.1.2 Tracking

Nachdem der Personendetektor eine Hypothese geliefert hat, ist es nun Aufgabe des Trackers die Hypothese zu verfolgen, auch bei einem kurzzeitigen Ausfall einer Erkennung durch den Personendetektor. Dazu muss der Tracker die Person zum Einen erkennen und zum Anderen seine Bewegungen über eine gewisse Zeit simulieren können. Schlussendliches Ziel ist es der roboterinternen Logik eine konstante Hypothese einer Person zu liefern.

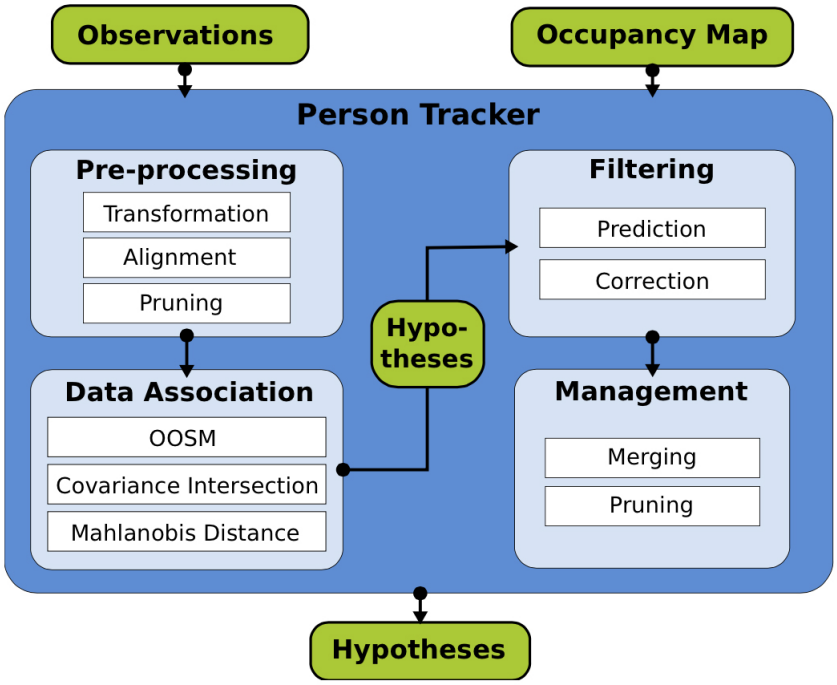

Abbildung 3.1: Übersicht des Person Tracker, Quelle: [Volkhardt et al., 2013]
Eine Übersicht der Schritte des Personen Trackings. Das pre-processing beschreibt die Personenerkennung des Person Tracker. Das eigentliche Tracking der Person findet in den Modulen Data Association, Filtering und Management statt.

Modelle

Der erste Schritt des Trackings ist das Filtern der durch die Personenerkennung gegebenen Hypothesen. Dieser Schritt ist das *Filtering* Modul in Abbildung 3.1. Aktuell wird in [VOLKHARDT et al., 2013] dafür ein linearer Kalman Filter mit einem 6-D Modell verwendet (FG-NIKR-Modell).

$$\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}) \tag{3.1}$$

Eine Person wird also im Raum durch seine Position und seine Geschwindigkeit in jede Raumrichtung beschrieben. Das führt zu folgendem Prädiktionsmodell:

$$\overline{\mathbf{x}}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & \triangle t & 0 & 0 \\ 0 & 1 & 0 & 0 & \triangle t & 0 \\ 0 & 0 & 1 & 0 & 0 & \triangle t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A_t}} \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \\ \dot{z}_{t-1} \end{pmatrix} = \begin{pmatrix} x_{t-1} & + & \dot{x}_{t-1}\triangle t \\ y_{t-1} & + & \dot{y}_{t-1}\triangle t \\ z_{t-1} & + & \dot{z}_{t-1}\triangle t \\ & & \dot{x}_{t-1} \\ & & \dot{y}_{t-1} \\ & & \dot{z}_{t-1} \end{pmatrix} \tag{3.2}$$

Als Beobachtung liefert der Personendetektor die Position einer Person im Raum.

$$\mathbf{z}_t = (x'_t, y'_t, z'_t) \tag{3.3}$$

Damit ergibt sich ein 3 x 6 Beobachtungsmodell:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{3.4}$$

Die Geschwindigkeiten der Raumrichtungen müssen also durch den Kalman Algorithmus gefiltert werden, da sie nicht gemessen werden.

Neben diesem verwendeten Modell soll auch das nichtlineare Modell aus [Bellotto und Hu, 2010] (Bellotto-Modell) im Rahmen dieser Arbeit integriert werden. Dieses Modell ist ein 5-D Modell, welches eine Person über seine räumliche x, y, z Position simuliert. Zusätzlich hat eine Person dann noch eine Orientierung ϕ in der $x-y$ Ebene und eine Geschwindigkeit v.

$$\mathbf{x} = (x, y, z, \phi, v) \tag{3.5}$$

Ein neuer Systemzustand wird dann folgendermaßen berechnet:

$$\overline{\mathbf{x}}_t = \begin{pmatrix} x_t \\ y_t \\ z_t \\ \phi_t \\ v_t \end{pmatrix} = \underbrace{\begin{pmatrix} x_{t-1} + v_{t-1}\triangle t\ cos\phi_{t-1} \\ y_{t-1} + v_{t-1}\triangle t\ sin\phi_{t-1} \\ z_{t-1} + n^z_{t-1} \\ \phi_{t-1} + n^\phi_{t-1} \\ |v_{t-1}| + n^v_{t-1} \end{pmatrix}}_{g(\mathbf{x})} \tag{3.6}$$

Dieses Modell ist somit ein nichtlineares Modell. n beschreibt einen Rauschterm, der auf z, ϕ und v addiert wird.
Als Messung soll wieder die Position der Person zum Tragen kommen. Somit muss vom Algorithmus die Geschwindigkeit v und die Orientierung ϕ gefiltert werden.

Tracker

Zusätzlich zum Filtern gibt es beim Personentracker in [Volkhardt et al., 2013] noch ein Hypothesen-Management, siehe *Management* Modul in Abbildung 3.1. Dieses führt folgende Schritte durch.

- Hypothesen mit ähnlicher Position und Geschwindigkeit werden zusammengeführt.
- Hypothesen mit einer hohen Kovarianz in den Ortskoordinaten werden entfernt.
- Detektionen in Wänden oder anderen Gegenständen und außerhalb der Karte werden entfernt. Dazu wird das Wissen des Roboters über seine Umwelt, z.B. über die Occupancy Map, genutzt.

Kalman Filter

Im aktuellem MIRA System sind Filter und Modelle immer fest verbunden. Ein implementierter Kalman Filter funktioniert also immer nur mit einem Modell. So ist es nicht möglich zur Laufzeit zwischen Filtern oder Modellen zu wechseln. Genutzt wird zur Zeit ein linearer Kalman Filter mit dem vorgestellen sechsdimensionalen Bewegungsmodell und ein Extended Kalman Filter (EKF) mit einem 9-dimensionalen Modell, [Weinrich et al., 2013].

Für das Personentracking können außerdem noch der in Kapitel 3 vorgestellte Unscented Kalman Filter und der Partikelfilter eingesetzt werden.

3.2 Kalman Filter Bibliotheken

Um die verschiedenen Kalman Filter und Modelle in der Softwareumgebung MIRA zu nutzen soll eine Bibliothek angebunden werden, die verschiedene Filter zur Verfügung stellt und eine einfach Implementierung der Modelle ermöglichen soll. Man findet dazu einige Alternativen, von denen im Folgenden eine Auswahl vorgestellt wird, angefangen mit der die für dieses Projekt verwendet wird.

3.2.1 Bayes++

Die Bayes++ Bibliothek von Michael Stevens vom "Australian Centre of Field Robotics" [Stevens, 2014] ist eine sehr umfangreiche Software Bibliothek. Geschrieben ist

sie in C++ und unter der MIT Lizenz[1] verfügbar. In der Bibliothek beinhaltet sind der Kalman, Extended Kalman und der Unscented Kalman Filter, sowie Partikel und Informationsfilter. Des Weiteren gibt es auch verschiedene Modellklassen, um ein System zu repräsentieren. Der große Vorteil der Bibliothek ist die Filter- bzw. Modellhierarchie. Diese macht es möglich, dass man mehrere Filter in der Softwareumgebung nutzen kann, die sich alle den selben Systemzustand teilen. Durch den großen Umfang und der Funktionsvielfalt der Bibliothek eignet sie sich sehr gut für das Personentracking.

3.2.2 Andere Bibliotheken

- KFilter [ZalZal, 2014]: Die KFilter Bibliothek implementiert nur den Extended Kalman Filter und ist deshalb nicht für dieses Projekt geeignet.
- Easykf [Eas, 2014]: Sowohl der Extended als auch der Unscented Kalman Filter werden hier implementiert. Die Bibliothek befinded sich jedoch noch im Aufbau.
- Orocos Bayesian Filtering Library [Gadeyne, 2001]: Auch hier wird wieder nur der Extended Kalman Filter und ein Partikelfilter angeboten.

[1] erlaubt dem Nutzer die Software frei zu verwenden, und für eigene Zwecke zu ändern

Kapitel 4

Anbindung der Bayes++ Bibliothek und Implementierung der Systemmodelle

Dieses Kapitel wird sich in zwei Teile gliedern. Im ersten Teil wird es um die Struktur und Anbindung der Bayes++ Bibliothek von [Stevens, 2014] an das MIRA-Framework gehen. Dabei wird kurz der Aufbau der Bibliothek besprochen, anschließend wird das Konzept für die Anbindung aufgezeigt und dabei die Funktionsweise erläutert. Im zweiten Teil wird dann die Implementierung des Modells aus [Bellotto und Hu, 2010] und die Implementierung des schon bestehenden Modells aus [Volkhardt et al., 2013] in das neue System besprochen.

4.1 Bayes++ Bibliothek

Wie schon kurz in Unterabschnitt 3.2.1 erläutert wurde, ist die Bayes++ Bibliothek eine umfangreiche Bibliothek für Bayes-Filter. Grundlage der Bibliothek sind die verschiedenen Filter- und Modellklassen, welche einen hierarchischen Aufbau haben. Trotz des komplexen Aufbaus gestaltet sich die Integration der Bibliothek recht simpel, da die API[1] bei allen Filter- und Modellklassen gleich ist. Diese Architektur

[1] application programming interface - Schnittstellen des Programms

wird über sogenannte Basisklassen, von denen sich alle anderen Klassen ableiten, realisiert. Dieser polymorphe Aufbau hat den großen Vorteil, dass man verschiedene Filter und Modelle zur Laufzeit nutzen kann. So können beliebige Filter-Modell-Kombinationen eingesetzt werden. Im Folgenden wird zuerst auf genau diesen Aufbau der Bibliothek näher eingegangen und anschließend die Anbindung an das MIRA Framework besprochen.

4.1.1 Aufbau

Der Aufbau der Bayes ++ Bibliothek teilt sich in zwei Bereiche, Filterklassen und Modellklassen. Beide Bereiche folgen jedoch dem selben Konzept des Polymorphismus. Durch Vererbung von Methoden und Variablen können in allen Klassen die selben Schnittstellen bereitgestellt werden. Dieses Konzept wird später auch für die Anbindung der Bibliothek genutzt.

In Abbildung 4.1 ist die Filterklassenstruktur der verwendeten Filterklassen der Bibliothek aufgezeigt.
Die Filter haben dabei vier Schnittstellen, welche für die Anbindung wichtig sind:

- **init:** Zum Initialisieren des Filters wird sein *init* aufgerufen, wobei Startzustand und Startkovarianz festgelegt werden.

- **update:** Über die *update* Methode wird der interne Filterzustand in eine Gauß-Form gebracht.

- **predict:** Der Prädiktionsschritt des Filters wird über *predict* getriggert. Dafür wird der Methode eine Referenz zu einem Prädiktionsmodell übergeben.

- **observe:** Für den Observationsschritt wird *observe* vom Tracker aufgerufen. Dafür benötigt die Methode wieder eine Referenz zu einem Observationsmodell und ein Beobachtungsvektor.

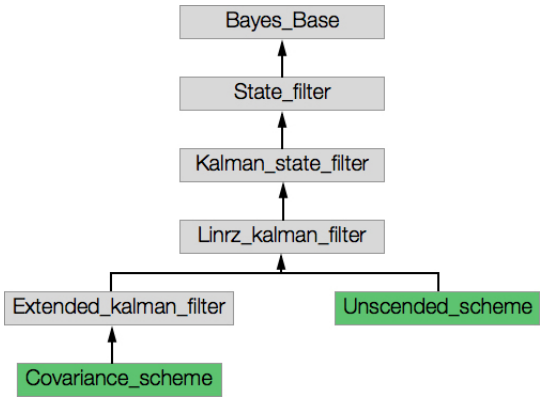

Abbildung 4.1: Filterklassenübsersicht der Bayes++ Bibliothek (Ausschnitt)
Die hier Grün dargestellten Klassen werden für die Anbindung der Bibliothek genutzt. Es ist zu sehen, dass beide Klassen der gleichen Basisklassenstruktur folgen. Die Basisklasse aller Kalman Filter ist die Kalman_ state_ filter Klasse. Bayes_ base ist die Basisklasse der gesamten Bibliothek.

Für jeden Filter werden immer zwei Modelle benötigt, ein Prädiktionsmodell und ein Observationsmodell. Auch hier gibt es verschiedene Klassen zu unterschiedlichen Modelltypen.

Genau wie bei den Filtern gibt es auch hier eine polymorphe Struktur, damit alle Modellklassen die selben Schnittstellen haben. Bei den Modellen wird jedoch zwischen Prädiktions- und Observationsmodell unterschieden. Ein Ausschnitt des Aufbaus der verwendeten Modellklassen ist in Abbildung 4.2 zu sehen.

Prädiktionsmodell:

- **f(Vektor x):** Über das Bewegungsmodell wird ein neuer Zustand geschätzt. Der Vektor **x** ist dabei der Zustand des Systems zum letzten Zeitschritt.

Observationsmodell:

- **h(Vektor x):** Es wird über den Vektor **x** aus dem Prädiktionsschritt ein Messvektor geschätzt.

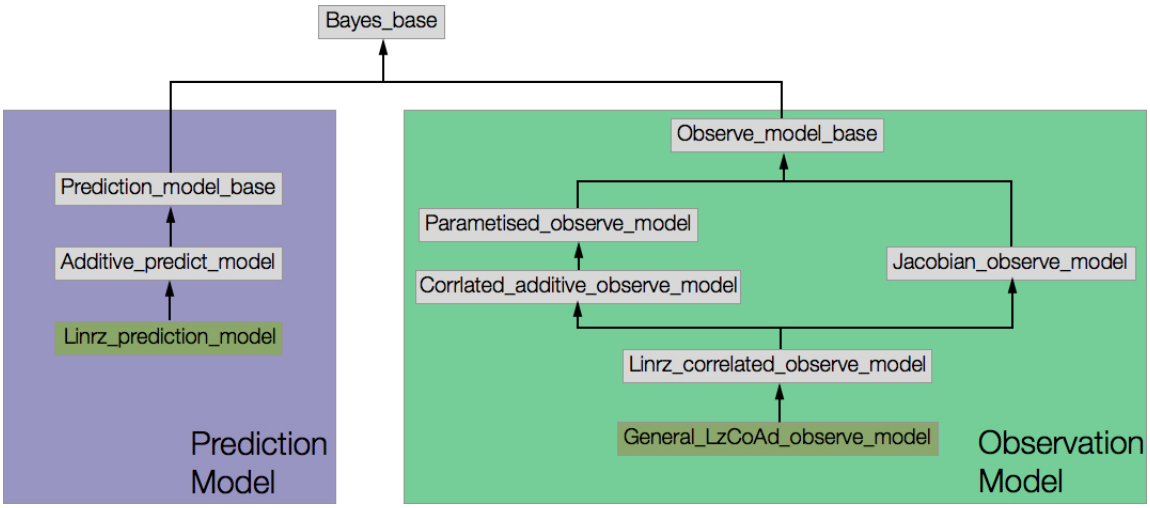

Abbildung 4.2: Modellklassenübersicht der Bayes++ Bibliothek (Ausschnitt)
Die verwendeten Klassen für die Anbindung des Prädiktions- und Observationsmodells sind Grün gekennzeichnet. Beide Klassen haben wieder eine Basisklasse, Prediction_model_base und Observe_model_base. Diese sind Ableitungen der Bibliotheksbasisklasse Bayes_base.

Über diese Schnittstellen muss nun die Bibliothek an das MIRA-Framework angebunden werden. Dies wird im nächsten Abschnitt besprochen.

4.1.2 Anbindung

Für die Anbindung an das MIRA Framework wurde, wie bei der Bayes++ Bibliothek, das Verfahren der Polymorphie und Vererbung genutzt, um eine dynamische Klassenstruktur zu erstellen. Diese Klassenstruktur muss dann zum Einen die Methoden der Bibliothek und zum Anderen die des Tracking-Moduls in MIRA implementieren.
Für die Filter ist es wichtig, dass dieser wiederum mit verschiedenen Modellen funktionieren. Dadurch ergab sich die in Abbildung 4.3 aufgezeigte Klassenstruktur.

Das Tracking findet in den beiden Klassen $ExtKalTracker$ und $UnsKalTracker$ statt. Die $ExtKalTracker$ Klasse implementiert dabei den Extended Kalman Filter und die $UnsKalTracker$ Klasse den Unscented Kalman Filter. Beide Klassen sind Ableitun-

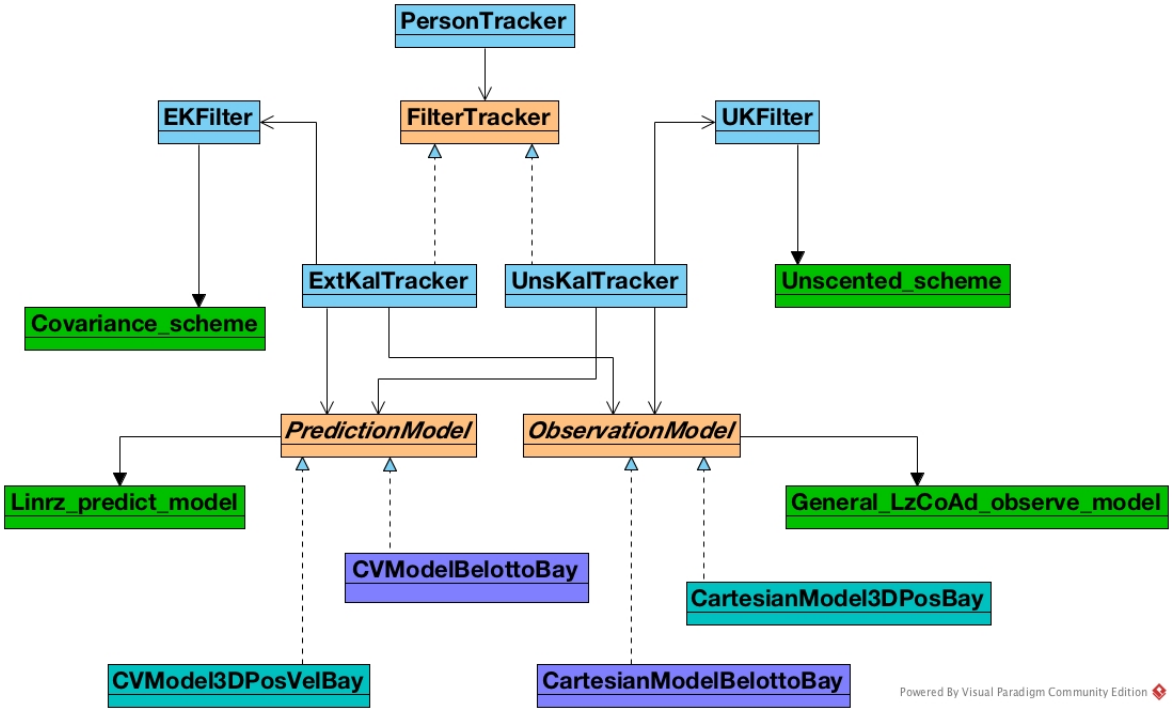

Abbildung 4.3: Vereinfachtes UML Diagramm der Implementierungsklassen

*In diesem vereinfachten Diagramm sind für die Implementierung alle notwendigen Klassen aufgeführt. Abstrakte Klassen sind dabei Orange und Klassen der Bayes++ Bibliothek sind Grün markiert. Die beiden Türkisen Modellklassen (unten) repräsentieren das Modell nach [*Volkhardt *et al., 2013] und die Lila gefärbten Klassen, im unteren Bildabschnitt, das Modell nach [*Bellotto *und* Hu*, 2010]. Gestrichelte Pfeile stellen eine Implementierung einer abstrakten Klasse dar und Pfeile mit ausgefülltem Kopf stehen für Ableitungen. Bei allen anderen Pfeilen handelt es sich um Propertys Beziehungen.*

gen der MIRA Klasse *FilterTracker* und bilden somit die Schnittstelle zum MIRA Framework.

Modellklassen

Beide Tracker haben als ihr Prädiktions- und Observationsmodell immer Objekte der Klassen *PredictionModel* und *ObservationModel*. Diese beiden abstrakten Klassen repräsentieren das Prädiktions-, bzw. Observationsmodell. Die eigentlichen Systemmodelle sind also immer Ableitungen dieser beiden Klassen. So ist es möglich ein beliebiges Systemmodell zu benutzten. Dazu kommt, dass die *PredictionModel* und *ObservationModel* Klassen Ableitungen von Bayes++ Modellklassen sind. Das ist notwendig, damit die Filter der Bayes++ Bibliothek diese Modelle benutzen können.

Prädiktionsmodell

Der Aufbau des Prädiktionsmodells ist in Abbildung 4.4 als UML Diagramm dargestellt.
Der Mittelpunkt der Abbildung 4.4 bildet die oben angesprochene abstrakte Klasse *PredictionModel*. Diese ist eine Ableitung der Bayes++ Klasse *Linrz_predict_model*, welche für lineare, oder linearisierte Modelle geeignet ist. Das eigentliche Bewegungsmodell ist dann eine Realisierung der *PredictionModel* Klasse, also die Klassen $CVModelBellottoBay$ (Bellotto-Modell) und $CVModel3DPosVelBay$ (FG-NIKR-Modell).

Das heißt, dass diese Klassen alle notwendigen Matritzen und Funktionen implementieren müssen, die von der *PredictionModel* Klasse vorgegeben werden und für den Prädiktionsschritt

$$\underbrace{\mathbf{x_t} = \mathbf{A_t}\mathbf{x_{t-1}} + \mathbf{B_t}\mathbf{u_t} + \boldsymbol{\varepsilon}_t}_{linear} \quad bzw. \quad \underbrace{\mathbf{x}_t = g(\mathbf{x}_{t-1}, \mathbf{u}_t) + \boldsymbol{\varepsilon}_t}_{nichtlinear} \tag{4.1}$$

notwendig sind.

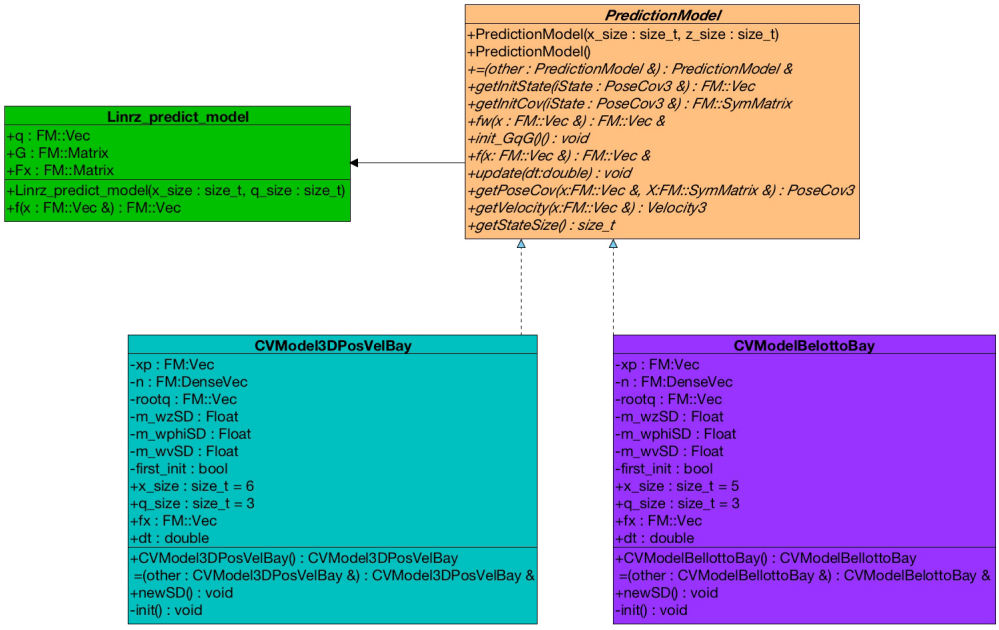

Abbildung 4.4: UML Diagramm der für das Prädiktionsmodell wichtigen Klassen *Die abstrakte Klasse PredictionModel ist selber eine Ableitung der Bayes++ Klasse Linrz_predict_model. Die beiden Klassen CVModelBelottoBay und CVModel3DPosVelBay realisieren die eigentlichen Bewegungsmodelle*

Das Bewegungsmodell $\mathbf{A_t}$, bzw. $g(\mathbf{x}_{t-1})$, wird dabei in einer Funktion $f(Vektorx)$ realisiert, während die Linearisierung als Matrix **Fx** implementiert ist.

- **f(Vector x)** - Implementierung des Bewegungsmodells, welches als Eingang den letzten Systemzustand erhält und die neue Schätzung des Systems als Rückgabewert hat.
- **Matrix Fx** - Implementierung der Jacobimatrix des Bewegungsmodells

Der additive Rauschterm des Prädiktionsschritts $\boldsymbol{\varepsilon_t}$ ist durch zwei Parameter definiert.

- **q** - Rauschvektor, welcher die Rauschparameter enthält
- **G** - Rauschmatrix, die die Rauschzusammenhänge implementiert

Die Parameter des Vektors $\mathbf{q}$ können dabei zur Laufzeit des Trackers geändert werden. Das endgültige Rauschen $\mathbf{Q}$ wird dann über $\mathbf{G}$ und $\mathbf{q}$ ausgerechnet. $\mathbf{Q}$ ist somit äquivalent zu $\boldsymbol{\varepsilon_t}$ aus Gleichung 4.1.2.

$$\mathbf{Q} = \mathbf{G} * diag(\mathbf{q}) * \mathbf{G}^T \tag{4.2}$$

Zusätzlich implementiert das Modell noch Funktionen, in denen der Systemzustand in eine Personenhypothese und umgekehrt, umgerechnet wird. Diese Umrechnung ist notwendig, da dieser Datentyp vom Filter an den Roboter weitergegeben wird.
Im derzeitigen MIRA Framework besteht eine solche Hypothese aus:

- Lage der Person - x, y, z
- Rotation der Person um alle Achsen - $yaw, pitch, roll$
- Kovarianzmatrix zur Position und Rotation
- Geschwindigkeit in jede Raumrichtung - v_x, v_y, v_z

Observationsmodell

Die *ObservationModel* Klasse ist ähnlich aufgebaut wie die *Predictionmodel* Klasse, siehe Abbildung 4.5.
Auch hier muss die eigentliche Beobachtungsmodellklasse wieder eine Ableitung der *ObservationModel* Klasse sein, welche selber eine Ableitung der bibliotheksinternen Klasse *General_LzCoAd_observe_model* ist. Diese Klasse wird für linearisierte Beobachtungsmodelle mit korrelierter Unsicherheit verwendet.
Auch hier wird der Observationsschritt des Kalman Filters

$$\underbrace{\mathbf{z_t} = \mathbf{C_t}\mathbf{x_t} + \boldsymbol{\delta_t}}_{linear} \quad bzw. \quad \underbrace{\mathbf{z_t} = h(\mathbf{x_t}) + \boldsymbol{\delta}_t}_{nichtlinear} \tag{4.3}$$

in den Realisierungen *CartesianModelBellottoBay* (Bellotto-Modell) und *CartesianModel3DPosBay* (FG-NIKR-Modell) durch Matrizen und Funktionen realisiert:

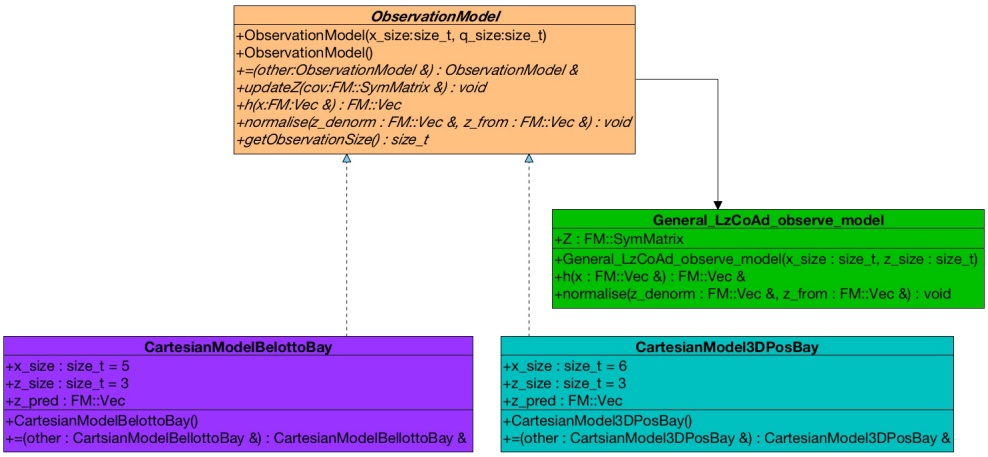

Abbildung 4.5: UML Diagramm der für das Observationsmodell wichtigen Klassen

Auch die ObservationModel Klasse ist abstrakt und wir durch die beiden Beobachtungsmodellklassen CartesianModelBellottoBay und CartesianModel3DPosBay realisiert. Sie wird abgeleitet von der Bayes++ Klasse General_LzCoAd_observe_model.

- **Matrix Hx** - Observationsmodell, welches die Messung auf den Systemzustand projiziert
- **Matrix Z** - Kovarianzmatrix der Messung
- **updateZ(Matrix cov)** - Projektion der Kovarianzmatrix der Messung auf die des Beobachtungsmodells
- **h(Vector x)** - Prädiktion des Messvektors mit dem aktuellen Systemzustand x

Durch die Matrix **Hx** kann sich ein Messvektor stark vom Systemzustand unterscheiden. Dadurch ist es möglich auch Messungen mit in die Observation zu integrieren, die nicht im Systemzustand repräsentiert sind.
Sind all diese Methoden und Matritzen implementiert ist das Systemmodell vollständig. Es besteht also aus zwei verschiedenen Klassen die Ableitungen der *PredictionModel* und *ObservationModel* Klasse sind.

Filterklassen

Zusätzlich zum Prädiktions- und Observationsmodell benötigt der Tracker noch eine Kalman Filter Klasse. Zum Einen gibt es die Klasse *EKFilter*, welche einen Extended Kalman Filter rerpäsentiert. Zum anderen wurde in *UKFilter* der Unscented Kalman Filter realisiert.

Extended Kalman Filter

Die Klasse *EKFilter* stellt durch ihre Ableitung der Klasse *Covariance_scheme* die Verbindung zur Bayes++ Bibliothek dar. Dieser Zusammenhang ist in Abbildung 4.6 nochmals dargestellt.

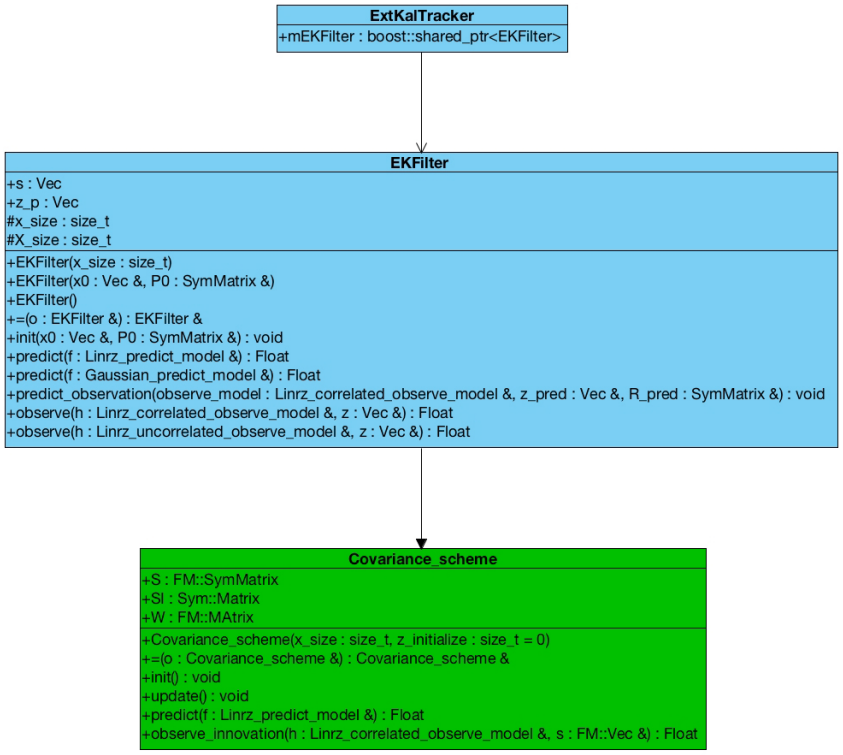

Abbildung 4.6: UML Diagramm der *EKFilter* Klasse
Die Klasse EKFilter ist eine Ableitung der Bayes++ Klasse Covariance_scheme, welche den Extended Kalman Filter implementiert. Die Trackerklasse ExtKalFilter hat immer ein EKFilter Objekt als Property.

Über die in Unterabschnitt 4.1.1 vorgestellten Schnittstellen *init*, *predict*, *observe* und *update* werden die bibliotheksinternen Abläufe aufgerufen. Die benötigten Modellreferenzen erhält der Filter dabei durch die Trackerklasse $ExtKalTracker$.

Unscented Kalman Filter

Um einen Unscented Kalman zu verwenden wurde die Klasse $UKFilter$ implementiert. Auch sie ist eine Ableitung einer Bibliotheksklasse, der Klasse $Unscented\_scheme$, welche diesmal einen Unscented Kalman Filter implementiert, siehe Abbildung 4.7.

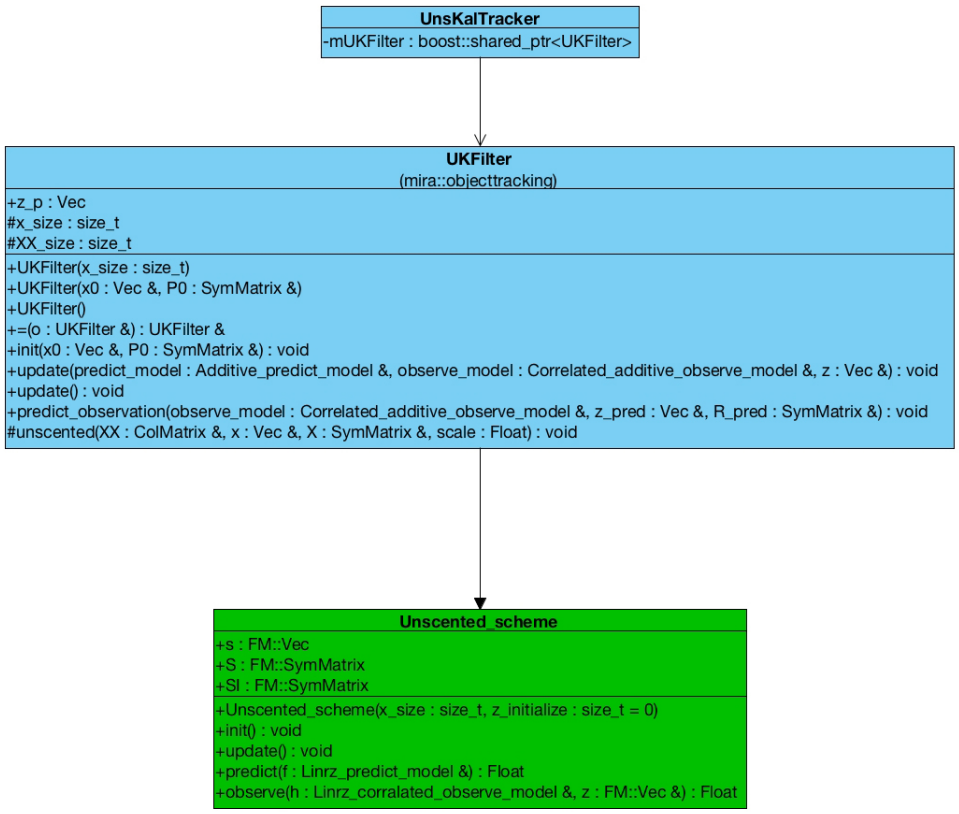

Abbildung 4.7: UML Diagramm der $UKFilter$ Klasse
Die $UKFilter$ Klasse ist eine Ableitung der Bayes++ Klasse $Unscented\_scheme$, welche einen Unscented Kalman Filter implementiert. Die Trackerklasse $UnsKalFilter$ hat immer ein $UKFilter$ Objekt als Property.

Wie beim *EKFilter* wird auch hier wieder die API der Bibliothek für die Kommunikation genutzt. Die Referenzen für das Prädiktions- und Observationsmodel werden hier durch die Klasse *UnsKalFilter* bereitgestellt.

Mit diesen beiden Klassen werden also die beiden Kalman Filter Variationen realisiert. Beide sind jeweils Ableitungen von Bayes++ Filterklassen deren gemeinsame Basisklasse die *Kalman_state_filter* Klasse ist, siehe Abbildung 4.1. Diese ist für die Verwaltung des Systemzustands verantwortlich. Hier wird die gemeinsame Zustandsrepräsentation für mehrere Filter integriert und sichergestellt, dass der Systemzustand immer eine Normalverteilung ist.

Trackerklassen

Die Trackerklassen *ExtKalTracker* und *UnsKalTracker* bilden das Bindeglied zwischen den vorgestellten Filter- und Modellklassen und dem MIRA Framework.

ExtKalTracker

Die *ExtKalTracker* Klasse repräsentiert ein Trackinginterface mit einem Extended Kalman Filter. Der Aufbau dieser Klasse ist in Abbildung 4.8 zu sehen.
Als Filter hat die Klasse ein Objekt der Klasse *EKFilter* als Property. Die beiden Modelle sind Objekte der abstrakten Klassen *Prediction−* und *ObservationModel*.
Die eigentlichen Modelle werden zur Laufzeit durch Serialisierung aus einer Konfigurationsdatei gelesen und integriert. Aktuell wird dafür in MIRA eine XML[2] Datei verwendet, in der die implementierten Modellklassen angegeben werden.
Dadurch, dass diese immer Ableitungen der abstrakten Modellklassen sind, kann ein beliebiges Modell mit dem Tracker verwendet werden. Aus diesen Modellen werden dann die für den Filter notwendigen Systemgrößen ausgelesen und an das *EKFilter* Objekt übergeben.

[2]XML steht für Extensible Markup Language und dient zur Darstellung hierarchisch strukturierter Daten in Textform

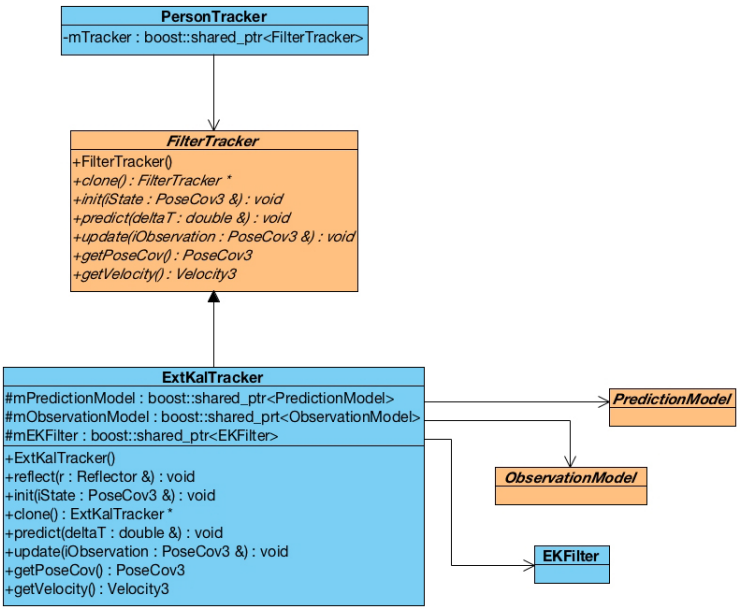

Abbildung 4.8: UML Übersicht der *ExtKalTracker* Klasse
Die ExtKalTracker Klasse ist eine Realisierung der abstrakten FilterTracker Klasse. Diese abstrakte Klasse ist ein Property der PersonTracker Klasse, welche die Grundklasse des Personen Trackers ist. Die ExtKalTracker Klasse hat selber als Propertys ein Objekt der EKFilter Klasse als Extended Kalman Filter und ein Prädiktions- und Observationsmodell, repräsentiert durch die PredictionModel und ObservationModel Klasse.

In der *ExtKalTracker* Klasse gibt es vier Funktionen, die für den Tracking-Ablauf wichtig sind.

- **init:** Es werden die Filterklasse und die Modelle initialisiert. Über eine Personenhypothese wird ein erster Systemzustand gebildet.
- **clone:** Es wird eine tiefe Kopie des Trackers erstellt.
- **predic:t(Time dt)** Der Prädiktionsschritt wird ausgeführt.
- **observe(Hypothese p):** Der Observationsschritt wird ausgeführt.

Der Inhalt der *predict* Methode ist in Abbildung 4.9 einmal dargestellt.

Eingaben

```
1   dt                                  // vergangene Zeit seit dem letzten Filterdurchlauf
```

Algorithmus

Predict: //

```
2       mPredictionModel.update( dt );          // Alle Zeitvarianten Matritzen im
Prädiktionsmodell werden angepasst
3       mFilter.predict( mPredictionModel );    // der Prädiktionsschritt der Bayes++
Bibliothek wird aufgerufen und das Prädiktionsmodell übergeben
4       mFilter.update();    // der Systeminterne Zustand wird wieder in eine Gauß-Form
gebracht
```

Abbildung 4.9: Predict-Methode der *ExtKalTracker* Klasse

Es wird zunächst in Zeile 2 die *update* Methode des Prädiktionsmodells aufgerufen. Dort werden alle zeitabhängigen Matrizen an die aktuelle Zeitdifferenz angepasst, z.B. die Jacobimatrix **Fx** oder die Rauschmatrix **Q**. Anschließend wird in Zeile 3 das *predict* des *EKFilter* aufgerufen und das aktualisierte *PredictionModel* Objekt als Referenz übergeben. Damit der interne Systemzustand wieder eine Gauß-Form hat wird in Zeile 4 noch die *update* Methode des Filters aufgerufen.

In Abbildung 4.10 ist der anschließende *observe* Schritt dargestellt. Dieser wird immer mit einer Observation aufgerufen. Daher werden in den Zeilen zwei bis fünf zunächst der für den Filter wichtige Observationsvektor und anschließend die Kovarianzmatrix der Observation aus der gegeben Observation gebildet.

In Zeile sieben wird dann die Kovarianzmatrix in das Observationsmodell gegeben, damit dieses seine interne Kovarianzmatrix zur aktuellen Observation anpassen kann. Danach wird dann in Zeile 8 das *observe* des Filters mit dem gebildeten Observationsvektor und dem *ObservationModel* aufgerufen. Auch hier wird danach wieder der interne Systemzustand in eine Gauß-Form, über die *update* Methode des Filters, gebracht.

Eingaben

```
1   mObservation                                    // Observation der Personenerkennung
```

Algorithmus

Observe: //

```
2   Vector tmpObsVector;                // Definition des temporären Beobachtungsvektors
3   tmpObsVector <- mObservation.position;   // Initialisierung des Vektors mit den
Positionsdaten der Beobachtung
4   Matrix tmpObsCovMatrix;                          // Definition der temporären
Beobachtungs-Kovarianzmatrix
5   tmpObsCovMatrix <- mObservation.positionCov;          // Initialisierung der
Kovarianzmatrix mit der Kovarianzmatrix der Beobachtung
6   mObservationModel.updateZ( tmpObsCovMatrix );         // Aktualisierung der
Observationsmodell internen Kovarianzmatrix mit der Kovarianzmatrix der Beobachtung
7   mFilter.observe(mObservationModel, tmpObsVector);  // Observationsschritt mit
Übergabe des Beobachtungsmodells und dem temporären Beobachtungsvektor
8   mFilter.update();     // der Systeminterne Zustand wird wieder in eine Gauß-Form
gebracht
```

Abbildung 4.10: Observe-Methode der *ExtKalTracker* Klasse

Mit diesen Schritten kann man mit dem *ExtKalTracker* nun eine Person mit einem Extended Kalman Filter der Bayes++ Bibliothek und einem belibigen Modell tracken.

UnsKalTracker

Die *UnsKalTracker* Klasse unterscheidet sich nur geringfügig von der *ExtKalTracker* Klasse. Wie in Abbildung 4.11 zu sehen ist, wird der Filter hier durch ein Objekt der *UKFilter* Klasse realisiert. Es wird also ein Unscented Kalman Filter verwendet.

Auch hier sind das Prädiktions- und Observationsmodell wieder durch die abstrakten Klassen *Prediction−* und *ObservationModel* realisiert. Somit können durch die Se-

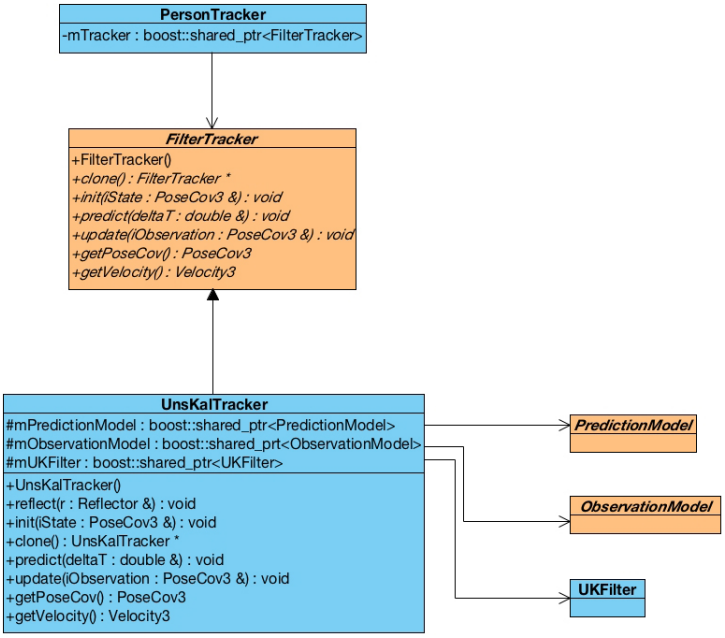

Abbildung 4.11: UML Übersicht der UnsKalTrackerKlasse

Die UnsKalTracker Klasse hat als Property immer ein Objekt der UKFilter Klasse, welche den Unscented Kalman Filter realisiert. Die Observations- und Prädiktionsmodelle werden durch die ObservationModel und PredictionModel Klassen realisiert. Die UnsKalTracker Klasse stellt selber eine Realisierung der abstrakten FilterTracker Klasse dar, welche die PersonTracker Klasse als Property besitzt. Diese Klasse ist die Kernklasse des Personentrackings in MIRA.

rialisierung wieder beliebige Modelle verwendet werden. Das Interface der Klasse ist analog zu dem der *ExtKalTracker* Klasse:

- **init:** Es werden die Filterklasse und die Modelle initialisiert. Über eine Personenhypothese wird ein erster Systemzustand gebildet.
- **clone:** Es wird eine tiefe Kopie des Trackers erstellt.
- **predic:t(Time dt)** Der Prädiktionsschritt wird ausgeführt.
- **observe(Hypothese p):** Der Observationsschritt wird ausgeführt.

Zusammenfassung

Mit den drei vorgestellen Elementen:

- **PredictionModel und ObservationModel** - Realisierung des Systemmodells
- **EKFilter und UKFilter** - Realisierung des Filteralgorithmus durch die Bayes++ Bibliothek
- **ExtKalTracker und UnsKalTracker** - Trackerklassen für die Zusammenführung und Verwaltung aller Elemente

kann man nun die Bayes++ Bibliothek mit einem Extended- und einem Unsceted Kalman Filter und das MIRA Framework verbinden. Dazu müssen nun nur noch geeignete Modelle implementiert werden, welches im nächsten Abschnitt beschrieben ist.

4.2 Implementierung der Systemmodelle

Wie in Abschnitt 2.4 schon beschrieben wurde, braucht man für einen Kalman Filter immer ein Systemmodell. In diesem Abschnitt wird nun die Implementierung von zwei verschiedenen Systemmodellen besprochen. Zum Einen soll das Modell aus [Bellotto und Hu, 2010] (Bellotto-Modell) implementiert werden und zum Anderen das schon in MIRA integrierte Modell aus [Volkhardt et al., 2013] (FG-NIKR-Modell) in das neues System überführt werden.

4.2.1 Modell nach Belotto

Das Modell nach [Bellotto und Hu, 2010] ist ein nichtlineares Modell mit einem 5-dimensionalen Zustand, welches in der Klasse $CVModelBellottoBay$ realisiert ist. Eine UML Übersicht dieser Klasse ist in Abbildung 4.12 dargestellt.

$$\mathbf{x} = (x, y, z, \phi, v) \tag{4.4}$$

Diese Beschreibung ist in Abbildung 4.13 noch einmal bildlich dargestellt.

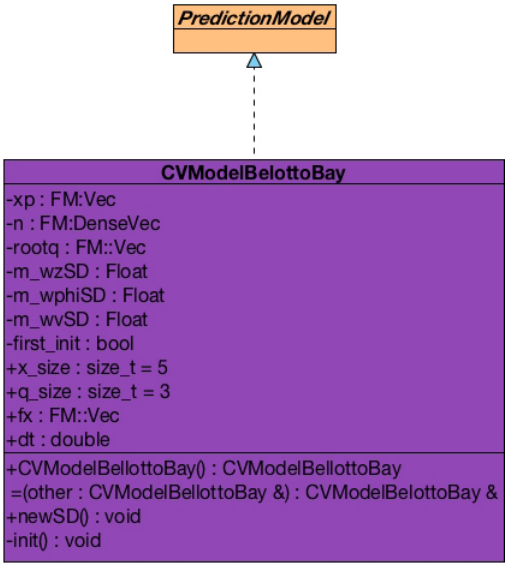

Abbildung 4.12: UML Übersicht der implementierten Prädiktionsmodellklasse *CVModelBellottoBay*

Die Klasse CVModelBellottoBay ist eine Realisierung der abstrakten PredictionModel Klasse.

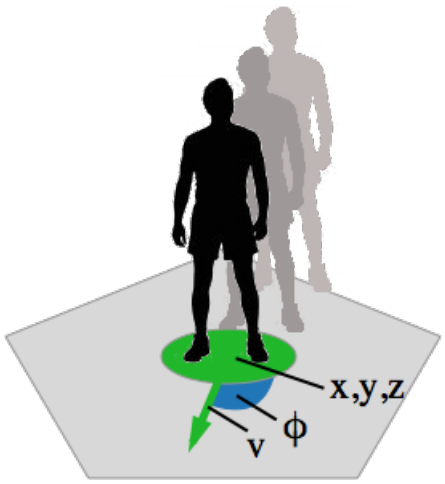

Abbildung 4.13: Systemmodel nach [Bellotto und Hu, 2010]

Die Bewegung einer Person wird durch seine Position, Geschwindigkeit und seiner Orientierung beschrieben.

Ein neuer Systemzustand wird dann folgendermaßen berechnet:

$$\overline{\mathbf{x}}_t = \begin{pmatrix} x_t \\ y_t \\ z_t \\ \phi_t \\ v_t \end{pmatrix} = \underbrace{\begin{pmatrix} x_{t-1} + v_{t-1}\triangle t\ cos\phi_{t-1} \\ y_{t-1} + v_{t-1}\triangle t\ sin\phi_{t-1} \\ z_{t-1} + n^z_{t-1} \\ \phi_{t-1} + n^\phi_{t-1} \\ |v_{t-1}| + n^v_{t-1} \end{pmatrix}}_{g(\mathbf{x})} \tag{4.5}$$

Das in Gleichung 4.5 dargestellte Bewegungsmodell wird in der Funktion $f(Vektor\ x)$ implementiert.

Für den Extended Kalman Filter muss jetzt dieser Funktionszusammenhang zu einer Jacobimatrix abgeleitet werden und in die Matrix **Fx** implementiert werden. Dazu wird jede Zeile nach jeder Variablen getrennt abgeleitet. Das führt zu folgender Linearisierung:

$$\mathbf{Fx} = \begin{pmatrix} 1 & 0 & 0 & -sin(\phi)\ v\ dt & cos(\phi)\ dt \\ 0 & 1 & 0 & cos(\phi)\ v\ dt & sin(\phi)\ dt \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & sgn(v) \end{pmatrix} \tag{4.6}$$

$sgn(x)$ ist dabei definiert als:

$$sgn(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{4.7}$$

Zu jedem Prädiktionsschritt gibt es immer noch ein Rauschterm $\mathbf{Q}$, welcher die Ungenauigkeit der Prädiktion beschreibt. Dieser ergibt sich aus dem Vektor $\mathbf{q}$ und einer Matrix $\mathbf{G}$.

$$\mathbf{Q} = \mathbf{G}\ diag(\mathbf{q})\ \mathbf{G}^T, mit\ \mathbf{q} = \begin{pmatrix} n^z \\ n^\phi \\ n^v \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.8}$$

Die Rauschterme n^z, n^ϕ und n^v werden erst zur Laufzeit initialisiert und können auch während des Trackings verändert werden. Sie beschreiben wie unsicher die Prädiktion eines neuen Zustands ist.
Mit diesen Zusammenhängen kann nun der Filter einen neuen Systemzustand schätzen.

Das Beobachtungsmodell ist in der Klasse *CartesianModelBellottoBay* aus Abbildung 4.14 implementiert.

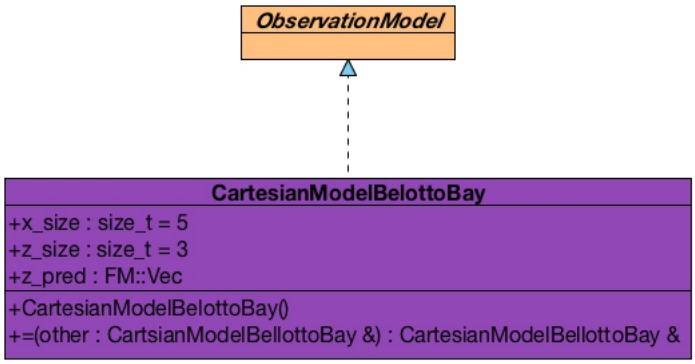

Abbildung 4.14: UML Ansicht der Implementierung des Beobachtungsmodells für das Bellotto-Modell, *CartesianModelBellottoBay*
Das Beobachtungsmodell wird durch die Klasse CartesianModelBellottoBay implementiert. Diese Klasse stellt eine Realisierung der abstrakten ObservationModel Klasse dar.

Als Beobachtung wird aktuell von der Personenerkennung in MIRA die Position der Person im Raum bereitgestellt. Das führt zu folgender Beobachtungsmatrix:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \textit{ mit Beobachtungsvektor } \quad \mathbf{z} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.9}$$

Als Kovarianzmatrix der Beobachtung wird die Kovarianzmatrix der Messung verwendet.

Mit diesen Zusammenhängen, die in den einzelen Klassen repräsentiert sind, kann nun dieses Modell mit der in Unterabschnitt 4.1.2 vorgestellten Implementierung der Bayes++ Bibliothek verwendet werden.

4.2.2 FG-NIKR-Modell

Das bisher verwendete Modell des Fachgebiets NIKR der TU-Ilmenau ist ein 6-dimensionales Modell, [Volkhardt et al., 2013]. Neben der Position einer Person im Raum wird im Systemzustand auch die Geschwindigkeit in jede Raumrichtung repräsentiert, siehe Abbildung 4.15. Das führt zu folgendem Systemzustand:

$$\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}) \tag{4.10}$$

Eine neuer Zustand ergibt sich also aus der alten Position und der Geschwindigkeit der Person in die Raumrichtungen.

$$\overline{\mathbf{x}}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & \triangle t & 0 & 0 \\ 0 & 1 & 0 & 0 & \triangle t & 0 \\ 0 & 0 & 1 & 0 & 0 & \triangle t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A_t}} \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \\ \dot{z}_{t-1} \end{pmatrix} = \begin{pmatrix} x_{t-1} + \dot{x}_{t-1}\triangle t \\ y_{t-1} + \dot{y}_{t-1}\triangle t \\ z_{t-1} + \dot{z}_{t-1}\triangle t \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \\ \dot{z}_{t-1} \end{pmatrix}$$

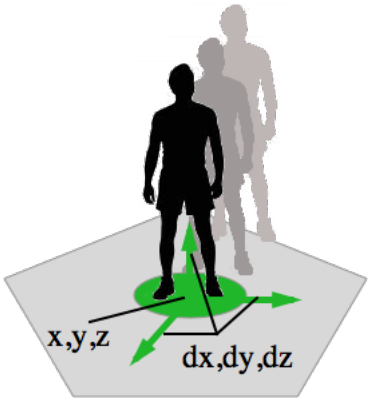

Abbildung 4.15: Systemmodell nach [Volkhardt et al., 2013]
Das Systemmodell besteht aus der Position der Person im Raum und der Geschwindigkeit in alle Raumrichtungen.

(4.11)

Dieses Bewegungsmodell wird durch die Klasse $CVModel3DPosVelBay$ aus Abbildung 4.16 implementiert.

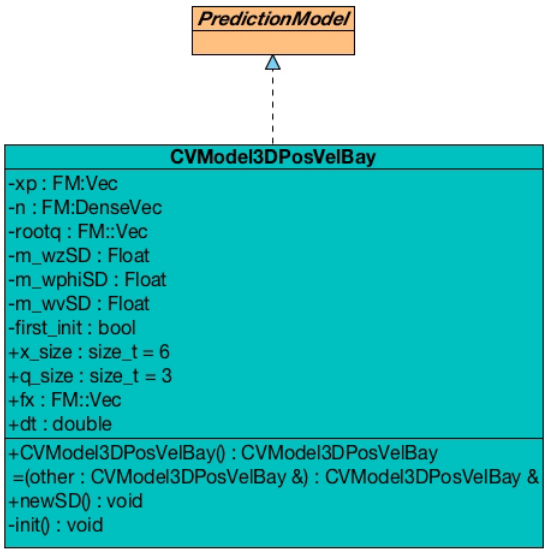

Abbildung 4.16: UML Ansicht der $CVModel3DPosVelBay$ Klasse
Diese Klasse implementiert das Bewegungsmodell aus [Volkhardt et al., 2013] durch eine Realisierung der abstrakten Klasse PredictionModel.

Das Bewegungsmodell aus Gleichung 4.11 wird auch hier in die Funktion $f(Vektor\ x)$ implementiert. Die Jacobimatrix der ersten Ableitung **Fx** ergibt sich dann folgendermaßen:

$$\mathbf{Fx} = \begin{pmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.12}$$

Der additive Rauschterm wird durch die unbekannte Beschleunigung simuliert, also ergibt sich mit den Gleichungen:

$$s = \frac{a}{2}t^2, \qquad v = at \tag{4.13}$$

die Matrix **G** und Vektor **q** wie folgt:

$$\mathbf{q} = \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} \frac{dt^2}{2} & 0 & 0 \\ 0 & \frac{dt^2}{2} & 0 \\ 0 & 0 & \frac{dt^2}{2} \\ dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \end{pmatrix} \tag{4.14}$$

Der Rauschterm ist hier also von der Zeit abhängig, wohingegen das Bellotto-Modell einen zeitunabhängigen Rauschterm hatte.

Das Beobachtungsmodell ist durch die Klasse $CartesianModel3DPosBay$ realisiert, siehe Abbildung 4.17.

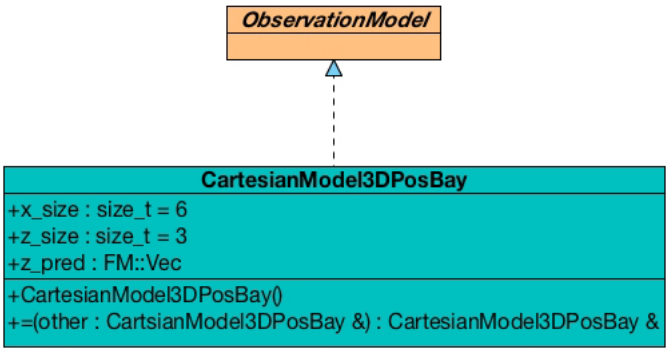

Abbildung 4.17: UML Ansicht der Klasse *CartesianModel3DPosBay*
Durch die Realisierung der abstrakten Klasse ObservationModel wird hier das Beobachtungsmodell des FG-NIKR-Modell's implementiert.

Die Beobachtung besteht auch hier aus der Position der Person. Damit ergibt sich die Beobachtungsmatrix:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \; mit \; Beobachtungsvektor \quad \mathbf{z} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.15}$$

Dadurch, dass nur die Position der Person beobachtbar ist, sind die Geschwindigkeiten in die Raumrichtungen das Ergebnis des Kalman Filter Algorithmus.

Mit diesen beiden Klassen kann nun auch das FG-NIKR-Modell in der neuen Tracker Umgebung verwenden.

4.2.3 Zusammenfassung

Es wurde gezeigt, wie die Bayes++ Bibliothek an das bestehende MIRA Framework angebunden wurde. Dafür wurden zwei verschiedene Trackingklassen beschrieben, die mit dem Personentracking System von MIRA genutzt werden können, die *ExtKalTracker* und *UnsKalTracker* Klasse. Diese beiden Trackerklassen haben jeweils ein Objekt einer Filterklasse als Property. Diese Filterklassen sind die *EKFilter*

und *UKFilter* Klasse und bilden die Verbindung zur Bayes++ Bibliothek, da sie von bibliotheksinternen Filterklassen ableiten. Als Prädiktions- und Observationsmodell wurden die *PredictionModel* und *ObservationModel* Klassen vorgestellt. Diese abstrakten Klassen haben Funktionen und Eigenschaften, die für die Verwendung in der MIRA Umgebung und für die Integration Bayes++ Bibliothek entscheidend sind. Die eigentlichen Systemmodelle müssen nun diese beiden Klassen realisieren und die Zusammenhänge der Systemmodelle implementieren. Diese Architektur wurde gewählt, damit man verschiedene Modelle über das Trackerinterface benutzen kann.

Die beiden Trackerklassen verbinden dann die Modelle mit dem Filter und führen die für die Filterung notwendigen Zwischenschritte durch, z.B. die Umrechnung der Beobachtung oder die Linearisierung des Bewegungsmodells.

Es wurden anschließend zwei Systemmodelle vorgestellt.
Das erste Modell nach [Bellotto und Hu, 2010] ist ein 5-dimensionales Model, welches neben den Raumkoordinaten noch die Orientierung in der x-y Ebene und die Geschwindigkeit einer Person als Zustandseigenschaft hat. Über diese beiden Zustandselemente wurde dann mit Hilfe von Winkelfunktionen eine neue Position einer Person prädiziert.
Das zweite Modell nach [Volkhardt et al., 2013] ist ein 6-dimensionales System, welche auch die Raumposition, sowie die Geschwindigkeit in jede Richtung einer Person beschreibt. Über die Geschwindigkeit in Kombination mit der Zeit wird dann ein neuer Zustand prädiziert.
Der Beobachtungsvektor für beide Modelle besteht aus den Raumkoordinaten der Person.

Das komplette System mit allen Klassen und deren Eigenschaften ist in Abbildung 4.18 nochmals als Gesamtübersicht zu sehen.
Nachdem sowohl die Bibliothek angebunden als auch die beiden Modelle in das neues System implementiert wurden, werden nun diese beiden Komponenten im folgenden Kapitel getestet und evaluiert.

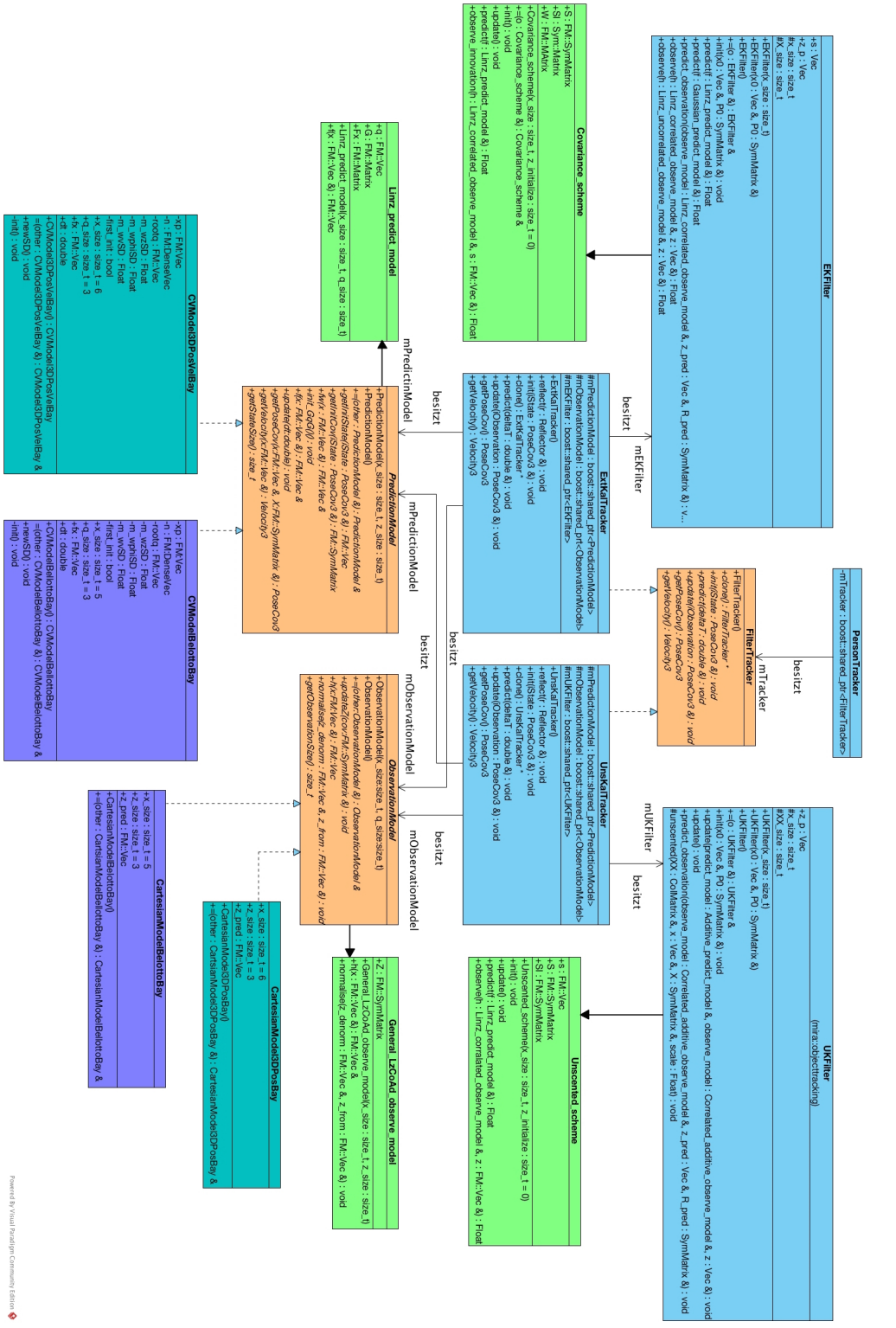

Abbildung 4.18: Gesamtansicht der Implementierung der Bayes++ Bibliothek, inklusive der Modelle nach [Bellotto und Hu, 2010] und [Volkhardt et al., 2013]

Kapitel 5

Evaluation

Die Evaluation der Bayes++ Bibliothek und der Bewegungsmodelle erfolgt in zwei Schritten. Zuerst wird überprüft, ob die Bibliothek dieselbe Performance liefert wie das bisher implementierte Trackingsystem. Danach werden die beiden Modelle über zwei Testverfahren miteinander verglichen.

5.1 Evaluierungsverfahren

Um verschiedene Tests vergleichen zu können benötigt man ein Evaluierungsverfahren, welches immer dieselben Randbedingungen stellt. Dies kann z.B. ein definierter Datensatz oder Datenabfolge sein. Hier werden sogenannte Tapes verwendet.
Sie sind Aufzeichnungen von Daten, welche von Sensoren oder Units[1] veröffentlicht wurden. Diese kann man dann offline wieder abspielen und so immer dieselbe Roboterfahrt simulieren.

Über diese Tapes wird zunächst die Bayes++ Bibliothek mit der aktuellen Kalman Filter Implementierung verglichen. Als Systemmodell wird das vorgestellte FG-NIKR-Modell benutzt.
Anschließend werden die zwei implementierten Bewegungsmodelle mit der neuen Bibliothek getestet. Jedes Modell wird dabei jeweils mit dem Extended Kalman Filter

[1] Unterprogramme in MIRA, z.B. Personentracking, Kollisionsvermeidung, Lokalisation, etc.

(EKF) und dem Unscented Kalman Filter (UKF) verwendet. Um die Performance der Modelle in verschiedenen Situationen zu testen werden 16 verschiedene Tapes verwendet. Diese beinhalten die für den Personendetektor und Tracker nötigen Daten und gelabelte Personenpositionen um den Tracker später auch bewerten zu können. Diese gelabelten Daten werden auch als *Ground Truth* Daten bezeichnet und simulieren das Ergebnis eines perfekten Trackers.

Als Bewertungsmaß wird der F-Score und die Multi-Object-Tracking-Performance aus [Bernardin et al., 2007] verwendet.
Der F-Score, oder F1-Score ist ein Maß, welches das harmonische Mittel der Genauigkeit (engl. *precision*) und der Trefferquote (engl. *recall*) bildet.

$$F = 2 \, \frac{precision \cdot recall}{precision + recall} \tag{5.1}$$

Precision und *recall* sind dabei Verhältnisse, welche Ergebnisse von Klassifikationsverfahren beschreiben. Gegeben ist dabei eine Grundmenge an Daten, welche in verschiedene Klassen geteilt werden soll, siehe Abbildung 5.1.
Precision ist das Verhältnis von allen richtigen Klassifikationen zu allen gemachten Klassifikationen. Recall ist dagegen das Verhältnis von allen richtig gemachten Klassifikationen zu allen möglich richtigen Klassifikationen der Daten.

Um dieses Bewertungsmaß nun auf den Personentracker anzuwenden muss die binäre Klassifikation auf das Trackingproblem übertragen werden. Dafür wird ein Posen[2]-Vergleich angewendet. Die Personenhypothese des Trackers wird mit der aufgenommenen Ground-Truth Pose des Tapes verglichen und die Distanz ausgerechnet. Ist diese klein genug, wird diese Hypothese als richtiges Ergebnis gewertet. Ist die Distanz allerdings zu groß, so wird dies als falsche "Klassifikation" gewertet, siehe Abbildung 5.2.

[2]Pose-Kurzform für Personenhypothese

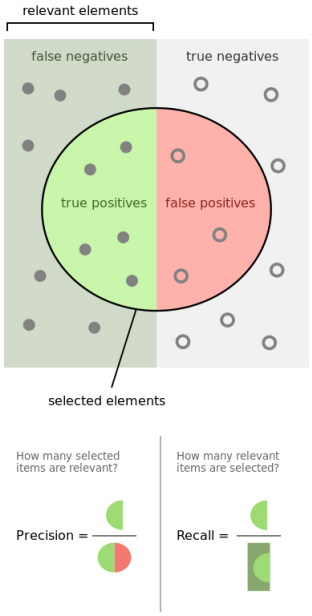

Abbildung 5.1: Precision und Recall, Quelle:[Walber, 2014]
Im oberen Bild sind alle Mengen von möglichen Beobachtungen dargestellt. Precision und Recall verrechnen dabei diese Mengen.

Als zweites Bewertungsmaß wird die $MOTP$ aus [Bernardin et al., 2007] verwendet. Die Multi-Object-Tracking-Performance ist ein Bewertungsverfahren, welches die Tracking Performance in zwei Werten beschreibt. Der erste Wert ist die Multiple Tracking Object Tracking Precision ($MOTP$) welche zeigt wie nah die Trackerposen an den aufgezeichneten des Tapes liegen.

$$MOTP = \frac{\Sigma_{i,t} d_{i,t}}{\Sigma_t c_c} \tag{5.2}$$

Dabei ist $d_{i,t}$ die Distanz zwischen der Trackerpose und der passenden $Ground-Truth$ Pose im Tape. c_t sind alle Posen des Trackers, die auch auf dem Tape gefunden wurden, sogenannte $Matches$.
Der zweite Wert ist die Multiple Object Tracking Accuracy ($MOTA$). Dieser Wert gibt an, wie gut eine Trajektorie eines Objektes gefolgt werden konnte. Dabei ist es wichtig, dass der Tracker immer denselben Menschen trackt und nicht auf einen ande-

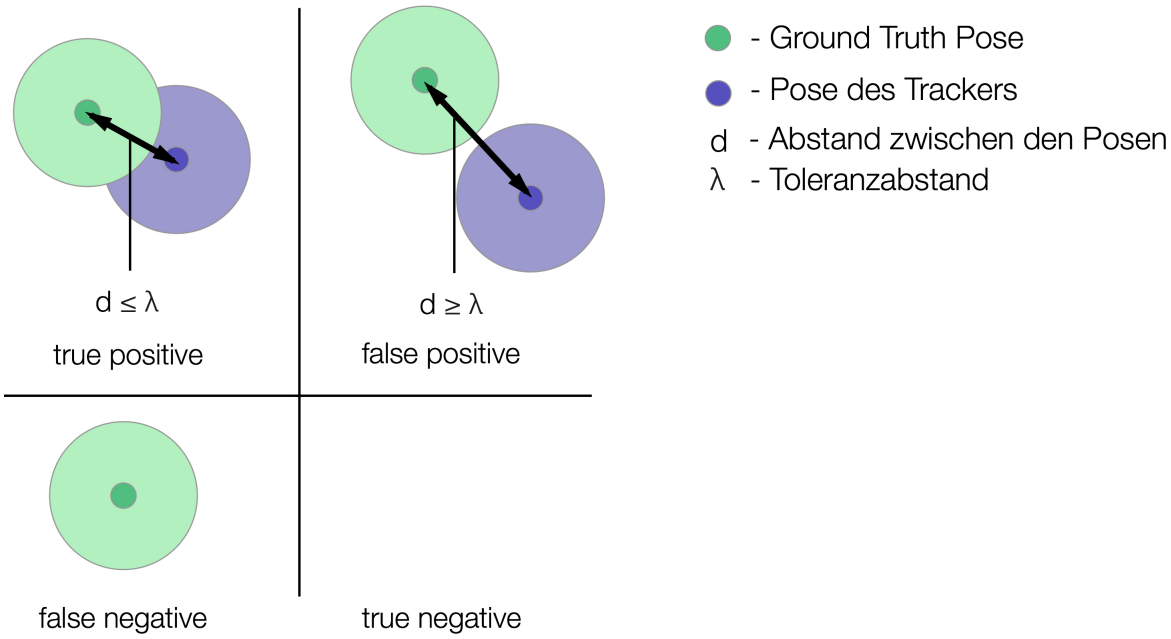

Abbildung 5.2: Posen-Interpretation einer binären Klassifikation
Eine Pose wird als true − positive gewertet, wenn der Abstand d kleiner ist als ein Maximalabstand λ (oben links). Ist der Abstand größer, so wird diese Pose als false − positive gewertet (oben rechts). Hat die Personenerkennung eine Person nicht erkannt, so ist dies ein false − negative Fall (unten links). Gibt es weder von der Personenerkennung noch in den Ground Truth Daten keine Pose, ist das ein true − negativ Fall (unten rechts).

ren überspringt. Dies ist häufig der Fall, wenn Personen nah aneinander vorbei laufen. Der $MOTA$ bestraft außerdem fehlende Tracking-Hypothesen und falsch-positive Hypothesen.

$$MOTA = 1 - \frac{\Sigma_t(m_t + fp_t + mme_t)}{\Sigma_t g_t} \tag{5.3}$$

Es wird also die Summe aus den fehlenden Hypothesen m_t, der falsch-positiven Hypothesen fp_t und der Vertauschungen mme_t in das Verhältnis zu allen Objekten über die gesamte Zeit gesetzt g_t.
Analog zum F-Score wird auch hier eine $true - positive$ Wertung über die Distanz der Tracker- und der Ground-Truth Pose gemacht.

Mit diesen Verfahren werden dann die beiden Modelle mit der Bibliotheksimplementierung getestet.

5.2 Testverfahren

Für ein vollständiges Testsystem wird neben den zu testenden Modellen und der Bibliothek noch eine Personentracker Umgebung benötigt. Dafür wird das in Abschnitt 3.1 vorgestellte System nach [Volkhardt et al., 2013] genutzt. Zur Personendetektion werden allerdings nur zwei Verfahren im Testsystem angewandt:

- **Bein-Detektor**: Über den Distanzlaser des Roboters können Beinpaare erkannt werden [Arras et al., 2007]. Ein Klassifikator trennt Beinpaare von anderen beinähnlichen Objekten, z.B. Tischbeinen.
- **Part HOG [Felzenszwalb et al., 2010]**: Der Part HOG nutzt dieselben Grundlagen wir der HOG, jedoch gibt es hier transformierbare, flexible Klassifikationsfenster, die auf das Bild angewendet werden. Damit können Personen auch in verschiedenen Posen und unter teilweiser Verdeckung erkannt werden.

Wie vorhin schon erwähnt wurde, werden Tapes genutzt, um eine vergleichbare Testumgebung zu schaffen. Dazu werden zum Beispiel Situationen im Haushalt simuliert, also in einer wohnungsähnlichen Umgebung mit teils laufenden und teils sitzenden Personen. Insgesamt kommen 16 Tapes zum Einsatz, deren Merkmale in Tabelle 5.1 aufgelistet sind.

Name	Länge	Merkmale
Tape 1	1:43	• mehrere Personen bewegen sich im Vorder- und Hintergrund • wechselnde Lichtverhältnisse
Tape 2	1:31	• Roboter folgt einer Person in einen Raum • andere Personen laufen zwischen Person und Roboter
Tape 3	4:07	• Roboter fährt durch die Wohnung und sucht Personen • Personen sitzen und sind teilweise durch Tische verdeckt
Tape 4	2:21	• sitzende Personen, die zum Teil verdeckt sind • Personen sitzen auf verschiedenen Möbel (Stuhl, Couch, Hocker)
Tape 5	4:41	• teils sitzende, teils sich bewegende Person
Tape 6	1:23	• dunkle Lichtverhältnisse • sitzende Person
Tape 7	3:43	• eine Person, die an verschiedenen Orten mal sitzt und mal steht
Tape 8	2:38	• eine Person auf verschiedenen Sitzgelegenheiten
chair	0:36	• Person sitzt auf einem Sessel
couch	0:45	• Person sitzt auf einer Couch an verschiedenen Positionen
follow	1:50	• Roboter folgt einer Person durch einen engen Gang
hallway	0:46	• Roboter steht und mehrere Personen bewegen sich vor dem Roboter
sitting 1	0:58	• Person sitzt auf Couch
sitting 2	0:38	• Person sitzt auf Couch
sitting 3	0:44	• Person sitzt auf Couch
sitting 4	1:17	• Person sitzt auf Hocker

Tabelle 5.1: Merkmale der verschiedenen Tapes

5.2.1 Bayes++ Bilbiothek

Für den Test der Bibliothek wird das 6-dimensionale Modell aus [Volkhardt et al., 2013] mit dem aktuellen Kalman Filter und mit dem EKF der neu implementierten Bayes++ Bibliothek verwendet. Als konstantes Rauschen des Modells wurden folgende Parameter aufgestellt:

$$\mathbf{q} = \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{5.4}$$

Die beiden Filtersysteme werden dann mit den ersten acht der in Tabelle 5.1 vorgestellten Tapes verwendet. Der Maximalabstand für eine korrekte Pose beträgt dabei $d = 0.5m$.
Die Ergebnisse sind in Abschnitt 5.3 aufgeführt.

5.2.2 Modelle

Neben der Implementierung der Bayes++ Bibliothek sollen auch die vorgestellten Systemmodelle, also das Bellotto-Modell und das FG-NIKR-Model, getestet werden. Für diesen Test wird ausschließlich die Bayes++ Bibliothek genutzt. Als Filter werden für beide Modelle ein EKF und ein UKF verwendet
Als Rauschterm für das Bellotto-Modell wird der Rauschvektor aus [Bellotto und Hu, 2010] verwendet:

$$\mathbf{q} = \begin{pmatrix} n^z \\ n^\phi \\ n^v \end{pmatrix} = \begin{pmatrix} 10^{-4} \\ \frac{\pi^2}{81} \\ 10^{-2} \end{pmatrix} \tag{5.5}$$

Für das FG-NIKR-Modell wird derselbe Rauschterm verwendet wie beim Test der Bibliothek.

$$\mathbf{q} = \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{5.6}$$

Die Bewertungsparameter des F-Score und des MOTP bleiben zum vorherigen Test auch gleich bei einem Maximalabstand von $d = 0.5m$ für eine korrekt getrackte Pose.

Für die Evaluierung werden beide Modelle mit jeweils dem UKF und dem EKF auf die 16 Tapes angewendet. Um ein sicheres Ergebnis bei allen Tapes zu erzielen, erfolgt dieser Schritt fünf mal hintereinander. Das eigentliche Ergebnis ist dann der Mittelwert der fünf Durchläufe. Durch die implementierte Bayes++ Bibliothek ist es auch nicht notwendig während der Tests Programmcode zu ändern, da man über ein Konfigurationsfile zur Laufzeit sowohl den Filter als auch das Modell festlegen kann. Während der Tracker mit dem Tape läuft, werden die Hypothesen die der Tracker berechnet, gespeichert und danach mit den im Tape gespeicherten Ground-Truth Hypothesen verglichen und der F-Score und MOTP Wert gebildet.

5.3 Ergebnisse und Auswertung

Bayes++ Bibliothek

Im ersten Test sollte gezeigt werden, dass die Bayes++ Bibliothek kein Leistungsverlust gegenüber dem aktuell implementierten Kalman Filter aufweist. Die Ergebnisse dieses Tests sind in Abbildung 5.3 aufgeführt.
Über alle Tapes verteilt haben beide Systeme sehr ähnliche Ergebnisse erzielt. Die auftretenden Differenzen bei einigen Tapes ist durch die Varianz der Ergebnisse zu erklären. Dies wird auch nocheinmal deutlich, wenn man sich die Mittlung über alle Tapes in Abbildung 5.4 anschaut, in denen beide Systeme die gleiche Ergebnisse aufweisen.

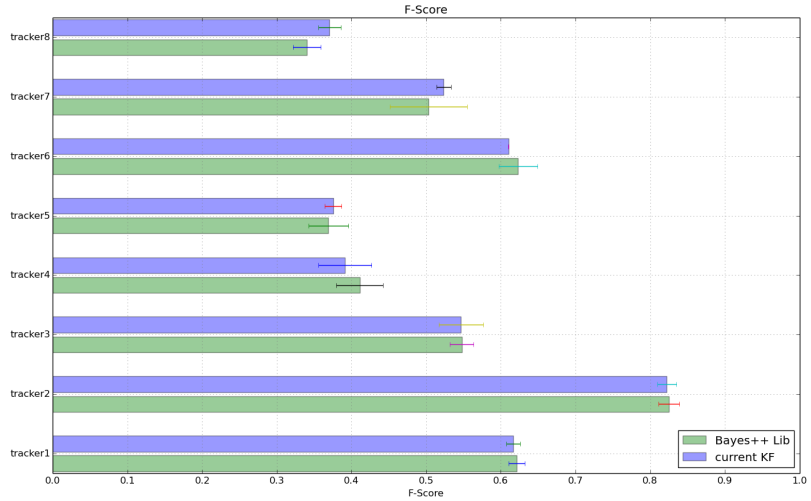

Abbildung 5.3: F-Score - Vergleich der Bayes++ Bibliothek, einzelne Tapes

Zu sehen sind die F-Score's der hier aufgeführten Tapes, welche einmal mit dem aktuell implementierten Kalman Filter (current KF), und einmal mit der Bayes++ Bibliothek (Bayes++ Lib) getestet wurden. Die Striche am Ende jedes Balkens markieren die Varianz des einzelnen Testergebnisses.

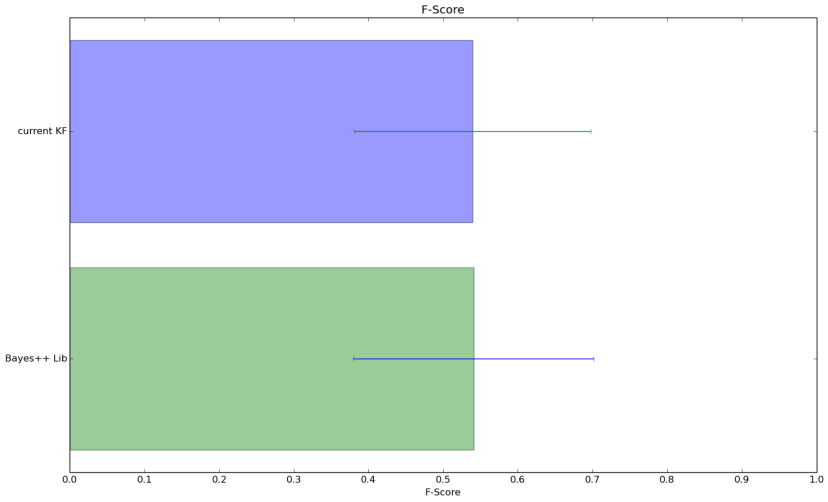

Abbildung 5.4: F-Score - Vergleich der Bayes++ Bibliothek, alle Tapes

Das hier aufgeführte Ergebnis ist die Mittlung über die in Abbildung 5.3 aufgeführten Tapes.

Systemmodelle

Im zweiten Teil der Evaluation sollten die beiden Systemmodelle verglichen werden. Dazu wurden die Modelle auf 16 Tapes mit mit dem F-Score und der MOTP verglichen.

Jedes Tape wurde mit jeweils einem der beiden Modelle getestet. Als Filter wurden der EKF und UKF eingesetzt. Jede Testkonfiguration wurde dann fünf mal verwendet, um aus diesen einen Durchschnittswert mit Varianz zu berechnen.

In Abbildung 5.5 und Abbildung 5.6 sind die F-Scores und die MOTP Werte der ersten 8 Tapes zu sehen.

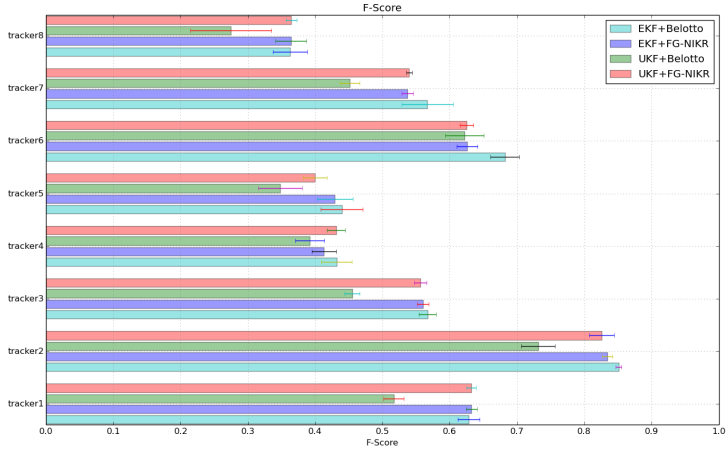

Abbildung 5.5: F-Score - Testergebnisse der Tapes 1-8
Zu sehen sind die F-Score's der hier aufgeführten Tapes mit den in der Legende aufgelisteten Filter-Modell-Kombinationen. Die Striche am Ende jedes Balkens markieren die Varianz des einzelnen Testergebnisses.

Bei allen Tapes ist ein ähnliches Verhalten zwischen Filter und Modell festzustellen. Verwendet man einen Extended Kalman Filter, unterscheiden sich die beiden Modelle kaum in ihrer Performance. Beim Unscented Kalman Filter schneidet jedoch das Bellotto-Modell bei allen Tapes schlechter ab als das FG-NIKR Modell. Es ist dabei zu

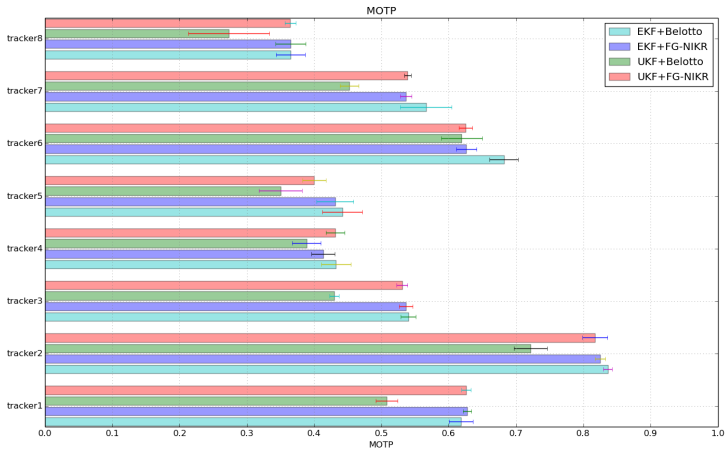

Abbildung 5.6: MOTP - Testergebnisse der Tapes 1-8
Zu sehen sind die MOTP Werte der hier aufgeführten Tapes mit den in der Legende aufgelisteten Filter-Modell-Kombinationen. Die Striche am Ende jedes Balkens markieren die Varianz des einzelnen Testergebnisses.

beachten, dass das aktuelle Modell linear ist, weshalb es kaum Unterschiede zwischen dem EKF und UKF gibt.
Das selbe Verhalten zeigt sich auch bei den restlichen 8 Tapes in Abbildung 5.7 und Abbildung 5.8.
Auch hier sieht man wieder einen deutlichen Unterschied zwischen dem UKF und EKF, wenn das Bellotto-Modell verwendet wurde. Außer bei dem Tape *sitting*2, wobei der UKF ein sehr viel besseres Ergebnis erzielen konnte. Bei allen anderen erreicht wieder der EKF mit beiden Modellen die besten Werte.

Bei allen 16 Tapes waren dementsprechend ähnliche Ergbnisse zu sehen. Dieses wird nocheinmal deutlich, wenn man sich die Mittlung über alle 16 Tapes in Abbildung 5.9 (F-Score) und Abbildung 5.10 (MOTP) anschaut. Auch hier sind die Werte beim EKF mit beiden Modellen und dem UKF mit dem Modell des FG-NIKR ähnlich. Nur der des Bellotto-Modells mit einen UKF schneidet schlechter ab.

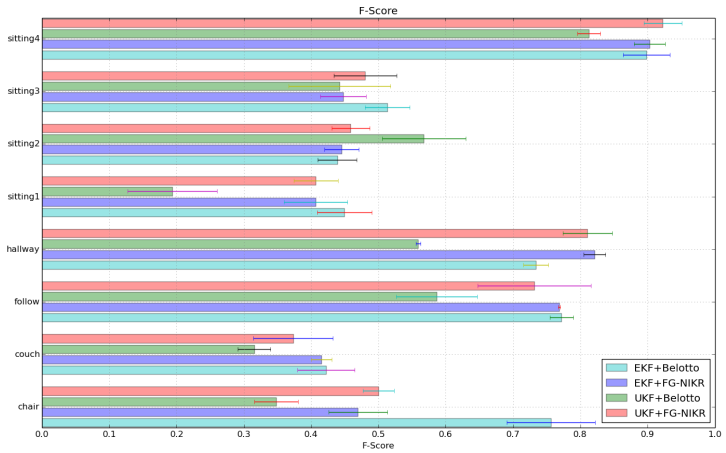

Abbildung 5.7: F-Score - Testergebnisse der Tapes 9-16

Zu sehen sind die F-Score's der hier aufgeführten Tapes mit den in der Legende aufgelisteten Filter-Modell-Kombinationen. Die Striche am Ende jedes Balkens markieren die Varianz des einzelnen Testergebnisses.

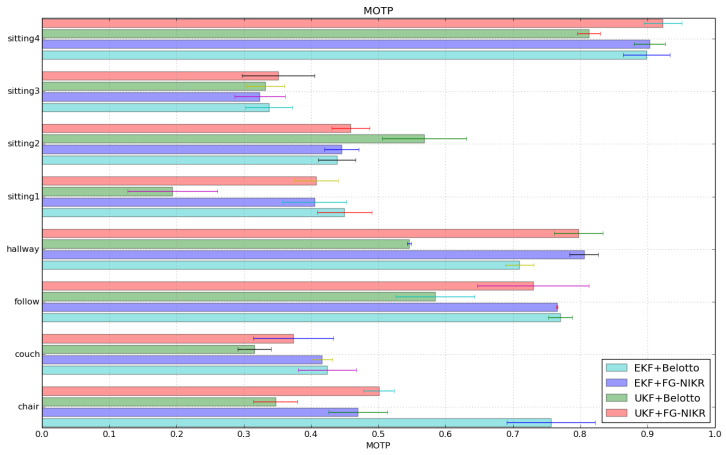

Abbildung 5.8: MOTP - Testergebnisse der Tapes 9-16

Zu sehen sind die MOTP Werte der hier aufgeführten Tapes mit den in der Legende aufgelisteten Filter-Modell-Kombinationen. Die Striche am Ende jedes Balkens markieren die Varianz des einzelnen Testergebnisses.

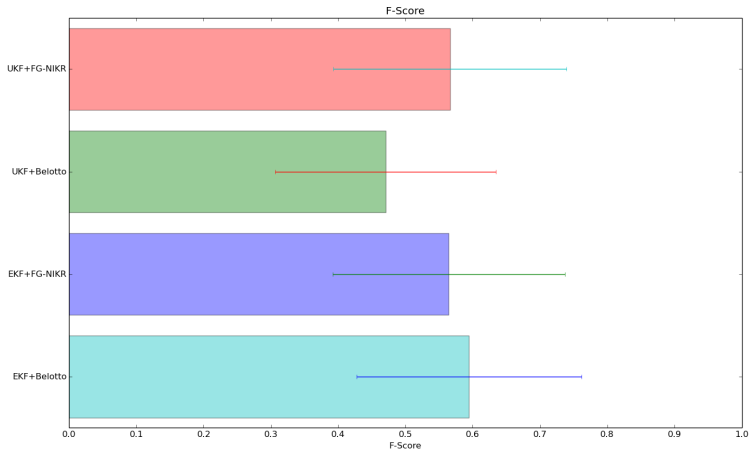

Abbildung 5.9: Mittlung aller F-Score Ergebnisse

Aufgeführt ist hier die Mittlung aller F-Score Ergebnisse der 16 Tapes. Die Varianz der einzelnen Konfigurationen wird durch die Striche am Ende der Balken dargestellt.

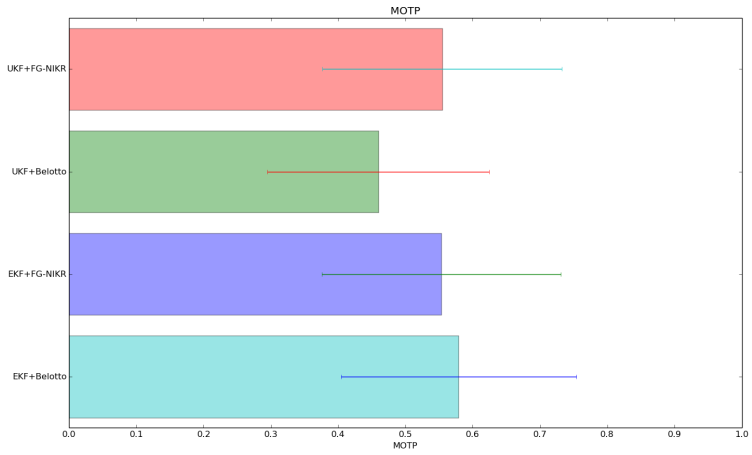

Abbildung 5.10: Mittlung aller MOTP Ergebnisse

Aufgeführt ist hier die Mittlung aller MOTP Ergebnisse der 16 Tapes. Die Varianz der einzelnen Konfigurationen wird durch die Striche am Ende der Balken dargestellt.

5.3.1 Vergleich der Systemmodelle

Obwohl es bei den Testergebnissen der beiden Modelle mit einem Extended Kalman Filter kaum Unterschiede gab, haben beide Modelle durch ihre unterschiedliche Systembeschreibung Vor- und Nachteile. Somit eignen sich die Modelle für verschiedene Situationen unterschiedlich gut.

Das lineare FG-NIKR Modell hat durch seine Systembeschreibung den Vorteil, dass die Bewegung einer Person sofort nach der Erkennung gut getrackt werden kann. Das gilt auch, wenn sich die Person zu Anfang des Trackings kaum bewegt. Dadurch, dass die Bewegung über die Geschwindigkeit in alle Raumrichtung simuliert wird, kann das Modell auf Bewegungen in alle Richtungen gleich schnell reagieren.
Dies ist zum Beispiel im Ergebnis des *hallway* Tape in Abbildung 5.7 zu sehen.
Das Modell eignet sich also für Situationen in denen Personen zum Teil stehen und sich bewegen, z.B. wenn sie den Roboter bedienen oder beim Start, wenn der Roboter einer Person folgen soll.

Das nichtlineare Bellotto-Modell hat den Vorteil Kurvenbewegungen von Personen gut zu simulieren. Beim Abspielen der Tapes war zu sehen, dass das Bellotto-Modell besser Personen folgen konnte als das FG-NIKR Modell, wenn diese Kurven gelaufen sind. Dadurch, dass die Rotation ein Teil des Systemzustands ist und davon ausgegangen wird, dass sich eine Person immer in Richtung dieser Orientierung läuft, reagiert das Modell besser auf Richtungsänderungen. Wie in Abbildung 5.11 zu sehen ist wird auch bei einer Kurvenbewegung sehr schnell wieder die richtige Orientierung der Person geschätzt.

Der Nachteil ist jedoch, dass es nach der Erfassung einer neuen Person eine Zeit lang dauert, bis die richtige Orientierung geschätzt wurde. Dadurch, dass das Modell mit einem Winkel von 0° initialisiert wird, benötigt der Filter einige Zeit bis der richtige Winkel der Bewegung getrackt wird.

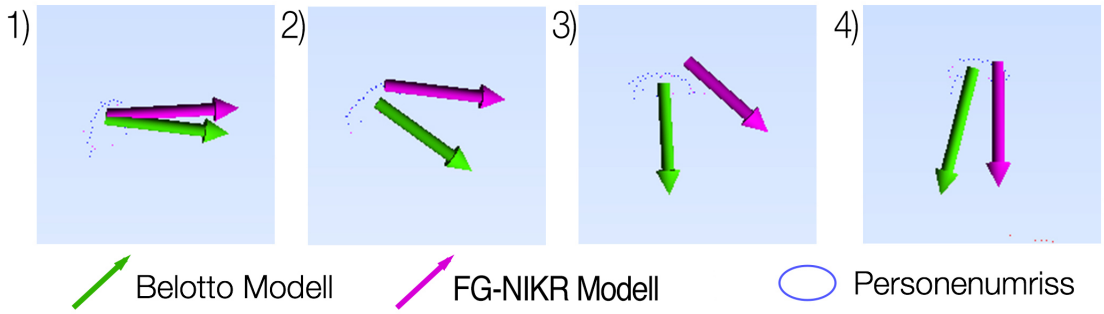

Abbildung 5.11: Verhalten der Modelle auf eine Rotationsbewegung einer Person *In Bild 1 (links) läuft die Person nach rechts und die Tracker beider Modelle zeigen in ungefähr die gleiche Richtung. In Bild 2 und 3 (links Mitte und rechts Mitte) macht die Person einen Rechtsbogen und man kann sehen, dass der Tracker mit dem Modell von Belotto sehr schnell seine Richtung anpasst. Dagegen ist beim FG-NIKR Modell zu erkennen, dass die Bewegung mit einer Verzögerung in das Trackingergebnis einfließt. Erst wenn die Person wieder einige Meter geradeaus gelaufen ist (Bild 4, rechts) gleichen sich die Richtungen der Tracker wieder an.*

Das Bellotto-Modell ist also für Situationen geeignet, bei der sich eine Person viel bewegt, z.B. bei Folge- oder Guiding-Aufgaben, wie im *follow* Tape in Abbildung 5.7.

5.4 Zusammenfassung

Insgesamt kann gesagt werden, dass die Bayes++ Bibliothek ohne Leistungsverlust gegenüber dem aktuellen System genutzt werden kann.
Außerdem ist es ohne Probleme möglich verschiedene Modelle und Filter mit der Bibliothek zu verwenden, wie in den Tests der Modelle gezeigt wurde.

Des Weiteren kann man sagen, dass, es mit einem Extended Kalman Filter wenig Unterschiede in der Trackingleistung beider Modelle gibt, obwohl die Systembeschreibungen recht unterschiedlich sind. Sowohl beim F-Score als auch beim MOTP zeigen beide Modelle ähnliche Ergebnisse. Natürlich sind beide Modelle damit nicht als gleich

zu bezeichnen, sondern besitzen beide Modelle ihre Vor- und Nachteile, welche sich jedoch in den verwendeten Tapes gegenseitig aufgehoben haben.
Mit der Funktionalität der Bibliothek kann jedoch durch die Verwendung mehrerer Modelle in verschiedenen Situationen die Vorteile dieser Modelle besser ausnutzen und so ein besseres Trackingergebnis erzielen.

Kapitel 6

Zusammenfassung und Ausblick

Ziel der Arbeit war es eine Filter Bibliothek an das MIRA-Framework anzubinden, welche es einfach machen sollte verschiedene Filter und Modell Kombinationen für den Personentracker zu verwenden. Außerdem sollte ein neues Systemmodell in diese Bibliothek implementiert und mit der aktuell implementierten Systembeschreibung einer Person verglichen werden.
Im gegebenen System waren alle Systemmodelle immer an einen bestimmten Filter gebunden und umgekehrt. So war es nur mit relativ großem Aufwand möglich neue Modelle und Filter in das System zu integrieren.

Um diese Probleme zu lösen wurde die in Kapitel 3 vorgestellte Bayes++ Bibliothek angebunden. In der Bibliothek sind alle benötigten Kalman Filter integriert und es ist durch den Aufbau der Bibliothek möglich, sowohl Filter als auch Modelle zur Laufzeit zu wechseln. Des Weiteren kann man ein implementiertes Modell mit verschiedenen Filtern anwenden und umgekehrt, so dass diese Bindung aufgehoben ist.

Als weiteres Systemmodell wurde ein 5-dimensionales Modell aus [Bellotto und Hu, 2010] verwendet, welches über die Position, Orientierung und Geschwindigkeit die Bewegung einer Person beschreibt und somit einen anderen Ansatz wählt, als das schon implementierte 6-dimensionale Modell aus [Volkhardt et al., 2013]. Dieses modelliert die Bewegung über die Position und die Geschwindigkeiten in die einzelnen

Raumrichtungen.

In Kapitel 4 wurde danach aufgezeigt, wie die Bayes++ Bibliothek an das MIRA-Framework angebunden wurde. Über teils abstrakte Zwischenklassen und Ableitungen dieser wurden die Architektur der Bibliothek mit dem der MIRA Umgebung verbunden. Anschließend wurden sowohl das Modell von Belotto als auch das Modell der Fachgebietes NIKR in das neue System integriert.

Getestet wurden diese Teile dann in Kapitel 5. Dort wurde zum Einen gezeigt, dass die neue Bibliothek richtig angebunden wurde und die beiden Modelle miteinander verglichen. Obwohl die beiden Systembeschreibungen unterschiedlich waren haben beide Modelle mit einem Extended Kalman Filter ähnliche Ergebnisse erzielt.
Bei den Tests der Modelle wurden auch gleich die Vorteile der Bayes++ Bibliothek genutzt und allein durch eine Konfigurationsdatei die einzelnen Filter- und Modell-Kombinationen ausgewählt.

Zusammenfassend kann gesagt werden, dass erfolgreich eine Bibliothek angebunden wurde, welche die gewünschten Funktionen besitzt. So können nun alle Vorteile der Bibliothek im MIRA-Framework genutzt werden.
Auch ein zweites Modell wurde erfolgreich implementiert und verglichen. Zwar gab es hier kaum Leistungsunterschiede, aber es wurden auch die Vorteile gegenüber dem FG-NIKR-Modell aufgezeigt.
Mit diesem System ist es nun möglich den Personentracker auf verschiedene Situationen anzupassen. So können verschiedene Modelle verwendet werden, wenn eine Person sitzt, steht oder sich bewegt, um die Trackingleistung des Roboters zu verbessern. Dazu müssen natürlich auch verschiedene Bewegungsmodelle entwickelt und getestet werden.
Ein Beispiel wären dabei sitzende Personen, bei denen sich ein Systemmodell eignet, welches eine statische Position simuliert, so dass die Person auch ohne Erkennung immer an der selben Stelle sitzt.
Neben dem Prädiktionsmodell ist es auch möglich nur das Observationsmodell in

einer bestimmten Situation zu wechseln. So könnte man zum Beispiel die Messung der Körperrotation, falls sie denn verfügbar ist, in den Messvektor integrieren.

Dadurch, dass die Bayes++ Bibliothek neben dem Kalman Filtern auch noch Informations- und Partikelfilter implementiert, kann sie auch außerhalb des Persontrackings eingesetzt werden.

Anhang A

Installation und Benutzung der Software

A.1 Installation

Nachdem das Softwarepaket *ObjectTracking* aus dem Repository des Fachgebiets für Neuroinformatik und Kognitive Robotik heruntergeladen wurde, muss dieses noch im Hauptprojektordner per *make* Befehl übersetzt werden.

Über das CMake-File des Projekts wird nun, bei bestehender Internetverbindung, die Bayes++ Bibliothek automatisch heruntergeladen und übersetzt.

Nachdem der Programmcode übersetzt wurde kann nun in der Konfigurationsdatei des *PersonTracker* als Filter die Klassen dieser Arbeit verwenden.

A.2 Verwendung

Im Folgenden ist eine Beispielkonfiguration für die Benutzung der Implementierung dieser Arbeit und damit der Bayes++ Bibliothek angegeben.

$< TrackerTemplate\ class = „mira :: objecttracking :: ExtKalTracker“ >$
$< PredictionModel\ class = „mira :: objecttracking :: CVModelBellottoBay“ >$
$< zDeviation > 0.0001 < /zDeviation >$
$< vDeviation > 0.02 < /vDeviation >$
$< phiDeviation > 0.5 < /phiDeviation >$
$< /PredictionModel >$
$< ObservationModel\ class = „mira :: objecttracking :: CartesianModelBellottoBay“ / >$
$< /TrackerTemplate >$

Als Erstes wird über den Tag *TrackerTemplate* die Tracker Klasse angegeben. Hier wurden in der Arbeit die *ExtKalTracker* und *UnsKalTracker* Klasse vorgestellt. Anschließend wird zuerst die Klasse des Prädiktionsmodells, inklusive der Rauschparameter und danach die des Beobachtungsmodells angegeben. Diese sollte aufeinander abgestimmt sein und demselben Systemmodell zugrunde liegen.
In diesem Beispiel wird der EKF mit dem Bellotto-Modell genutzt. Deshalb wurde als TrackerTemplate die Klasse *ExtKalTracker*, als Prädiktionsmodell *CVModelBellottoBay* und als Observationsmodell *CartesianModelBellottoBay* eingesetzt.
Die beim *PredictionModel* angegeben Parameter können auch später im *miracenter* verändert werden. Es ist jedoch zu beachten, dass diese erst bei einer neuen Personenhypothese wirksam werden.

Anhang B

Detallierte Testergebnisse

Im Folgenden sind zu jeder Grafik aus Kapitel 5 nochmals die Zahlenwerte der Tests angegeben.

Abbildung 5.3: F-Score - Vergleich der Bayes++ Bibliothek, einzelne Tapes:

$tracker1.tape:$	$Bayes++Lib:$	$\mu = 0.621 \quad \sigma : 0.011$
	$currentKF:$	$\mu = 0.617 \quad \sigma : 0.009$
$tracker2.tape:$	$Bayes++Lib:$	$\mu = 0.825 \quad \sigma : 0.014$
	$currentKF:$	$\mu = 0.822 \quad \sigma : 0.013$
$tracker3.tape:$	$Bayes++Lib:$	$\mu = 0.548 \quad \sigma : 0.016$
	$currentKF:$	$\mu = 0.547 \quad \sigma : 0.03$
$tracker4.tape:$	$Bayes++Lib:$	$\mu = 0.411 \quad \sigma : 0.031$
	$currentKF:$	$\mu = 0.391 \quad \sigma : 0.035$
$tracker5.tape:$	$Bayes++Lib:$	$\mu = 0.369 \quad \sigma : 0.026$
	$currentKF:$	$\mu = 0.375 \quad \sigma : 0.011$
$tracker6.tape:$	$Bayes++Lib:$	$\mu = 0.623 \quad \sigma : 0.026$
	$currentKF:$	$\mu = 0.61 \quad \sigma : 0.001$
$tracker7.tape:$	$Bayes++Lib:$	$\mu = 0.503 \quad \sigma : 0.052$
	$currentKF:$	$\mu = 0.523 \quad \sigma : 0.01$
$tracker8.tape:$	$Bayes++Lib:$	$\mu = 0.34 \quad \sigma : 0.019$
	$currentKF:$	$\mu = 0.371 \quad \sigma : 0.015$

Abbildung 5.5: F-Score - Testergebnisse der Tapes 1-8:

$tracker1.tape:$	$EKF + Belotto:$	$\mu = 0.628 \quad \sigma: 0.016$
	$EKF + FG - NIKR:$	$\mu = 0.632 \quad \sigma: 0.008$
	$UKF + Belotto:$	$\mu = 0.517 \quad \sigma: 0.015$
	$UKF + FG - NIKR:$	$\mu = 0.632 \quad \sigma: 0.007$
$tracker2.tape:$	$EKF + Belotto:$	$\mu = 0.851 \quad \sigma: 0.005$
	$EKF + FG - NIKR:$	$\mu = 0.834 \quad \sigma: 0.008$
	$UKF + Belotto:$	$\mu = 0.731 \quad \sigma: 0.025$
	$UKF + FG - NIKR:$	$\mu = 0.826 \quad \sigma: 0.019$
$tracker3.tape:$	$EKF + Belotto:$	$\mu = 0.567 \quad \sigma: 0.013$
	$EKF + FG - NIKR:$	$\mu = 0.56 \quad \sigma: 0.008$
	$UKF + Belotto:$	$\mu = 0.455 \quad \sigma: 0.011$
	$UKF + FG - NIKR:$	$\mu = 0.556 \quad \sigma: 0.009$
$tracker4.tape:$	$EKF + Belotto:$	$\mu = 0.432 \quad \sigma: 0.023$
	$EKF + FG - NIKR:$	$\mu = 0.413 \quad \sigma: 0.018$
	$UKF + Belotto:$	$\mu = 0.392 \quad \sigma: 0.022$
	$UKF + FG - NIKR:$	$\mu = 0.431 \quad \sigma: 0.014$
$tracker5.tape:$	$EKF + Belotto:$	$\mu = 0.44 \quad \sigma: 0.031$
	$EKF + FG - NIKR:$	$\mu = 0.429 \quad \sigma: 0.027$
	$UKF + Belotto:$	$\mu = 0.348 \quad \sigma: 0.032$
	$UKF + FG - NIKR:$	$\mu = 0.4 \quad \sigma: 0.018$
$tracker6.tape:$	$EKF + Belotto:$	$\mu = 0.682 \quad \sigma: 0.021$
	$EKF + FG - NIKR:$	$\mu = 0.626 \quad \sigma: 0.015$
	$UKF + Belotto:$	$\mu = 0.622 \quad \sigma: 0.028$
	$UKF + FG - NIKR:$	$\mu = 0.625 \quad \sigma: 0.01$
$tracker7.tape:$	$EKF + Belotto:$	$\mu = 0.567 \quad \sigma: 0.039$
	$EKF + FG - NIKR:$	$\mu = 0.537 \quad \sigma: 0.008$
	$UKF + Belotto:$	$\mu = 0.451 \quad \sigma: 0.015$
	$UKF + FG - NIKR:$	$\mu = 0.54 \quad \sigma: 0.005$
$tracker8.tape:$	$EKF + Belotto:$	$\mu = 0.363 \quad \sigma: 0.026$
	$EKF + FG - NIKR:$	$\mu = 0.364 \quad \sigma: 0.023$
	$UKF + Belotto:$	$\mu = 0.274 \quad \sigma: 0.06$
	$UKF + FG - NIKR:$	$\mu = 0.364 \quad \sigma: 0.008$

Abbildung 5.6: MOTP - Testergebnisse der Tapes 1-8:

$tracker1.tape:$	$EKF + Belotto:$	$\mu = 0.618 \quad \sigma: 0.018$
	$EKF + FG - NIKR:$	$\mu = 0.627 \quad \sigma: 0.006$
	$UKF + Belotto:$	$\mu = 0.508 \quad \sigma: 0.016$
	$UKF + FG - NIKR:$	$\mu = 0.626 \quad \sigma: 0.007$
$tracker2.tape:$	$EKF + Belotto:$	$\mu = 0.837 \quad \sigma: 0.007$
	$EKF + FG - NIKR:$	$\mu = 0.825 \quad \sigma: 0.008$
	$UKF + Belotto:$	$\mu = 0.722 \quad \sigma: 0.025$
	$UKF + FG - NIKR:$	$\mu = 0.818 \quad \sigma: 0.018$
$tracker3.tape:$	$EKF + Belotto:$	$\mu = 0.54 \quad \sigma: 0.011$
	$EKF + FG - NIKR:$	$\mu = 0.536 \quad \sigma: 0.01$
	$UKF + Belotto:$	$\mu = 0.43 \quad \sigma: 0.007$
	$UKF + FG - NIKR:$	$\mu = 0.531 \quad \sigma: 0.008$
$tracker4.tape:$	$EKF + Belotto:$	$\mu = 0.432 \quad \sigma: 0.022$
	$EKF + FG - NIKR:$	$\mu = 0.413 \quad \sigma: 0.018$
	$UKF + Belotto:$	$\mu = 0.389 \quad \sigma: 0.021$
	$UKF + FG - NIKR:$	$\mu = 0.431 \quad \sigma: 0.014$
$tracker5.tape:$	$EKF + Belotto:$	$\mu = 0.442 \quad \sigma: 0.03$
	$EKF + FG - NIKR:$	$\mu = 0.431 \quad \sigma: 0.027$
	$UKF + Belotto:$	$\mu = 0.35 \quad \sigma: 0.032$
	$UKF + FG - NIKR:$	$\mu = 0.4 \quad \sigma: 0.017$
$tracker6.tape:$	$EKF + Belotto:$	$\mu = 0.682 \quad \sigma: 0.021$
	$EKF + FG - NIKR:$	$\mu = 0.626 \quad \sigma: 0.015$
	$UKF + Belotto:$	$\mu = 0.619 \quad \sigma: 0.031$
	$UKF + FG - NIKR:$	$\mu = 0.625 \quad \sigma: 0.01$
$tracker7.tape:$	$EKF + Belotto:$	$\mu = 0.566 \quad \sigma: 0.038$
	$EKF + FG - NIKR:$	$\mu = 0.537 \quad \sigma: 0.008$
	$UKF + Belotto:$	$\mu = 0.452 \quad \sigma: 0.014$
	$UKF + FG - NIKR:$	$\mu = 0.539 \quad \sigma: 0.005$
$tracker8.tape:$	$EKF + Belotto:$	$\mu = 0.365 \quad \sigma: 0.022$
	$EKF + FG - NIKR:$	$\mu = 0.365 \quad \sigma: 0.022$
	$UKF + Belotto:$	$\mu = 0.273 \quad \sigma: 0.06$
	$UKF + FG - NIKR:$	$\mu = 0.364 \quad \sigma: 0.008$

Abbildung 5.7: F-Score - Testergebnisse der Tapes 9-16:

chair.tape :	$EKF + Belotto$:	$\mu = 0.757 \quad \sigma : 0.066$
	$EKF + FG - NIKR$:	$\mu = 0.469 \quad \sigma : 0.043$
	$UKF + Belotto$:	$\mu = 0.348 \quad \sigma : 0.032$
	$UKF + FG - NIKR$:	$\mu = 0.5 \quad \sigma : 0.023$
couch.tape :	$EKF + Belotto$:	$\mu = 0.422 \quad \sigma : 0.043$
	$EKF + FG - NIKR$:	$\mu = 0.415 \quad \sigma : 0.016$
	$UKF + Belotto$:	$\mu = 0.315 \quad \sigma : 0.025$
	$UKF + FG - NIKR$:	$\mu = 0.373 \quad \sigma : 0.059$
$follow.tape$:	$EKF + Belotto$:	$\mu = 0.772 \quad \sigma : 0.017$
	$EKF + FG - NIKR$:	$\mu = 0.769 \quad \sigma : 0.001$
	$UKF + Belotto$:	$\mu = 0.587 \quad \sigma : 0.06$
	$UKF + FG - NIKR$:	$\mu = 0.732 \quad \sigma : 0.084$
hallway.tape :	$EKF + Belotto$:	$\mu = 0.734 \quad \sigma : 0.019$
	$EKF + FG - NIKR$:	$\mu = 0.821 \quad \sigma : 0.016$
	$UKF + Belotto$:	$\mu = 0.559 \quad \sigma : 0.004$
	$UKF + FG - NIKR$:	$\mu = 0.811 \quad \sigma : 0.036$
*sitting*1.*tape* :	$EKF + Belotto$:	$\mu = 0.449 \quad \sigma : 0.041$
	$EKF + FG - NIKR$:	$\mu = 0.407 \quad \sigma : 0.047$
	$UKF + Belotto$:	$\mu = 0.193 \quad \sigma : 0.066$
	$UKF + FG - NIKR$:	$\mu = 0.407 \quad \sigma : 0.033$
*sitting*2.*tape* :	$EKF + Belotto$:	$\mu = 0.439 \quad \sigma : 0.029$
	$EKF + FG - NIKR$:	$\mu = 0.445 \quad \sigma : 0.026$
	$UKF + Belotto$:	$\mu = 0.567 \quad \sigma : 0.062$
	$UKF + FG - NIKR$:	$\mu = 0.458 \quad \sigma : 0.028$
*sitting*3.*tape* :	$EKF + Belotto$:	$\mu = 0.513 \quad \sigma : 0.033$
	$EKF + FG - NIKR$:	$\mu = 0.447 \quad \sigma : 0.034$
	$UKF + Belotto$:	$\mu = 0.442 \quad \sigma : 0.076$
	$UKF + FG - NIKR$:	$\mu = 0.48 \quad \sigma : 0.047$
*sitting*4.*tape* :	$EKF + Belotto$:	$\mu = 0.898 \quad \sigma : 0.035$
	$EKF + FG - NIKR$:	$\mu = 0.903 \quad \sigma : 0.023$
	$UKF + Belotto$:	$\mu = 0.813 \quad \sigma : 0.017$
	$UKF + FG - NIKR$:	$\mu = 0.923 \quad \sigma : 0.028$

Abbildung 5.8: MOTP - Testergebnisse der Tapes 9-16:

chair.tape :	$EKF + Belotto$:	$\mu = 0.757 \quad \sigma : 0.066$
	$EKF + FG - NIKR$:	$\mu = 0.469 \quad \sigma : 0.043$
	$UKF + Belotto$:	$\mu = 0.347 \quad \sigma : 0.033$
	$UKF + FG - NIKR$:	$\mu = 0.5 \quad \sigma : 0.023$
couch.tape :	$EKF + Belotto$:	$\mu = 0.423 \quad \sigma : 0.043$
	$EKF + FG - NIKR$:	$\mu = 0.416 \quad \sigma : 0.016$
	$UKF + Belotto$:	$\mu = 0.315 \quad \sigma : 0.025$
	$UKF + FG - NIKR$:	$\mu = 0.373 \quad \sigma : 0.059$
$follow.tape$:	$EKF + Belotto$:	$\mu = 0.77 \quad \sigma : 0.018$
	$EKF + FG - NIKR$:	$\mu = 0.765 \quad \sigma : 0.001$
	$UKF + Belotto$:	$\mu = 0.584 \quad \sigma : 0.059$
	$UKF + FG - NIKR$:	$\mu = 0.73 \quad \sigma : 0.083$
hallway.tape :	$EKF + Belotto$:	$\mu = 0.71 \quad \sigma : 0.021$
	$EKF + FG - NIKR$:	$\mu = 0.806 \quad \sigma : 0.021$
	$UKF + Belotto$:	$\mu = 0.546 \quad \sigma : 0.003$
	$UKF + FG - NIKR$:	$\mu = 0.797 \quad \sigma : 0.036$
*sitting*1.*tape* :	$EKF + Belotto$:	$\mu = 0.449 \quad \sigma : 0.041$
	$EKF + FG - NIKR$:	$\mu = 0.405 \quad \sigma : 0.047$
	$UKF + Belotto$:	$\mu = 0.194 \quad \sigma : 0.067$
	$UKF + FG - NIKR$:	$\mu = 0.407 \quad \sigma : 0.032$
*sitting*2.*tape* :	$EKF + Belotto$:	$\mu = 0.438 \quad \sigma : 0.028$
	$EKF + FG - NIKR$:	$\mu = 0.445 \quad \sigma : 0.026$
	$UKF + Belotto$:	$\mu = 0.568 \quad \sigma : 0.063$
	$UKF + FG - NIKR$:	$\mu = 0.458 \quad \sigma : 0.028$
*sitting*3.*tape* :	$EKF + Belotto$:	$\mu = 0.337 \quad \sigma : 0.035$
	$EKF + FG - NIKR$:	$\mu = 0.324 \quad \sigma : 0.037$
	$UKF + Belotto$:	$\mu = 0.332 \quad \sigma : 0.029$
	$UKF + FG - NIKR$:	$\mu = 0.351 \quad \sigma : 0.054$
*sitting*4.*tape* :	$EKF + Belotto$:	$\mu = 0.899 \quad \sigma : 0.035$
	$EKF + FG - NIKR$:	$\mu = 0.903 \quad \sigma : 0.023$
	$UKF + Belotto$:	$\mu = 0.813 \quad \sigma : 0.017$
	$UKF + FG - NIKR$:	$\mu = 0.923 \quad \sigma : 0.028$

Abbildung 5.4: F-Score - Vergleich der Bayes++ Bibliothek, alle Tapes:

$Bayes++Lib:$	$\mu = 0.541 \quad \sigma = 0.161$
$currentKF:$	$\mu = 0.539 \quad \sigma = 0.158$

Abbildung 5.9: Mittlung aller F-Score Ergebnisse:

$EKF + Belotto:$	$\mu = 0.595 \quad \sigma = 0.167$
$EKF + FG - NIKR:$	$\mu = 0.564 \quad \sigma = 0.173$
$UKF + Belotto:$	$\mu = 0.471 \quad \sigma = 0.164$
$UKF + FG - NIKR:$	$\mu = 0.566 \quad \sigma = 0.174$

Abbildung 5.10: Mittlung aller MOTP Ergebnisse:

$EKF + Belotto:$	$\mu = 0.579 \quad \sigma = 0.175$
$EKF + FG - NIKR:$	$\mu = 0.553 \quad \sigma = 0.178$
$UKF + Belotto:$	$\mu = 0.46 \quad \sigma = 0.165$
$UKF + FG - NIKR:$	$\mu = 0.555 \quad \sigma = 0.178$

Abbildungsverzeichnis

Literaturverzeichnis

[Jul, 1997] (1997). *New extension of the Kalman filter to nonlinear systems*, Bd. 3068.

[Eas, 2014] (2014). *EasyKF - C++ Kalman Filtering*. `http://jeremyfix.github.io/easykf/`.

[Arras et al., 2007] Arras, K.O., O. Mozos und W. Burgard (2007). *Using Boosted Features for the Detection of People in 2D Range Data*. In: *Robotics and Automation, 2007 IEEE International Conference on*, S. 3402–3407.

[Bellotto und Hu, 2010] Bellotto, Nicola und H. Hu (2010). *Computationally efficient solutions for tracking people with a mobile robot: an experimental evaluation of Bayesian filters*. Autonomous Robots, 28(4):425–438.

[Bernardin et al., 2007] Bernardin, Keni, T. Gehrig und R. Stiefelhagen (2007). *Multi-and Single View Multiperson Tracking for Smart Room Environments*. In: Stiefelhagen, Rainer und J. Garofolo, Hrsg.: *Multimodal Technologies for Perception of Humans*, Bd. 4122 d. Reihe *Lecture Notes in Computer Science*, S. 81–92. Springer Berlin Heidelberg.

[Dalal und Triggs, 2005] Dalal, N. und B. Triggs (2005). *Histograms of oriented gradients for human detection*. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Bd. 1, S. 886–893 vol. 1.

[Dollar et al., 2010] Dollar, Piotr, S. Belongie und P. Perona (2010). *The Fastest Pedestrian Detector in the West*. In: *Proceedings of the British Machine Vision Conference*, S. 68.1–68.11. BMVA Press.

[Einhorn et al., 2012] Einhorn, E., T. Langner, R. Stricker, C. Martin und H. Gross (2012). *MIRA - middleware for robotic applications*. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, S. 2591–2598.

[Felzenszwalb et al., 2010] Felzenszwalb, P.F., R. Girshick und D. McAllester (2010). *Cascade object detection with deformable part models*. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, S. 2241–2248.

[Gadeyne, 2001] Gadeyne, Klaas (2001). *BFL: Bayesian Filtering Library*. `http://www.orocos.org/bfl`.

[Goehler, 2007] Goehler, w. (2007). *Formelsammlung HÃ¶here Mathematik*. Verlag Harri Deutsch, Frankfurt am Main.

[Gross et al., 2011] Gross, H., C. Schroeter, S. Mueller, M. Volkhardt, E. Einhorn, A. Bley, C. Martin, T. Langner und M. Merten (2011). *Progress in developing a socially assistive mobile home robot companion for the elderly with mild cognitive impairment*. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, S. 2430–2437.

[Gross et al., 2014] Gross, H.-M., K. Debes, E. Einhorn, S. Mueller, A. Scheidig, C. Weinrich, A. Bley und C. Martin (2014). *Mobile Robotic Rehabilitation Assistant for walking and orientation training of Stroke Patients: A report on work in progress*. In: *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, S. 1880–1887.

[Julier und Uhlmann, 2004] Julier, S.J. und J. Uhlmann (2004). *Unscented filtering and nonlinear estimation*. Proceedings of the IEEE, 92(3):401–422.

[Kalman, 1960] Kalman, R. E. (1960). *A New Approach to Linear Filtering And Prediction Problems*. ASME Journal of Basic Engineering.

[Stevens, 2014] Stevens, Michael (2014). *Bayexs++ Library*. `http://bayesclasses.sourceforge.net/Bayes++.html`.

[Stricker et al., 2012] Stricker, R., S. Muller, E. Einhorn, C. Schroter, M. Volkhardt, K. Debes und H. Gross (2012). *Interactive mobile robots guiding visitors in a university building*. In: *RO-MAN, 2012 IEEE*, S. 695–700.

[Thrun et al., 2006] Thrun, S., W. Burgard und D. Fox (2006). *Probalistic Robotics*. MIT Press, Cambridge.

[Viola und Jones, 2004] Viola, Paul und M. Jones (2004). *Robust Real-Time Face Detection*. International Journal of Computer Vision, 57(2):137–154.

[Volkhardt et al., 2013] Volkhardt, M., C. Weinrich und H.-M. Gross (2013). *People Tracking on a Mobile Companion Robot*. In: *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, S. 4354–4359.

[Walber, 2014] Walber (2014). *Precision and recall*. `http://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg`.

[Weinrich et al., 2013] Weinrich, C., M. Volkhardt, E. Einhorn und H.-M. Gross (2013). *Prediction of human collision avoidance behavior by lifelong learning for socially compliant robot navigation*. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, S. 376–381.

[Weiprecht, 2012] Weiprecht, Juergen (2012). *Wahrscheinlichkeiten, Wahrscheinlichkeitsverteilungen*. `http://www.astro.uni-jena.de/Teaching/Praktikum/pra2002/node276.html`.

[Welch und Bishop, 1995] Welch, Greg und G. Bishop (1995). *An Introduction to the Kalman Filter*. Technischer Bericht, Chapel Hill, NC, USA.

[ZalZal, 2014] ZalZal, Vincent (2014). *KFilter - Free C++ Extended Kalman Filter Library*. `http://kalman.sourceforge.net/index.php`.